A Novel Developmental Genetic Programming Methodology for Mathematical Modeling and Neuroevolution

by

Stephen Johns

B.Sc. Ryerson University, 2007

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Canada, 2010

Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signed:_____

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signed:_____

A Novel Developmental Genetic Programming Methodology for Mathematical Modeling and Neuroevolution

Stephen Johns

M. Sc. in Computer Science, 2010

Ryerson University, Toronto, Canada

Abstract

In this work, a novel developmental genetic programming methodology called NEXT (Next Encoding of eXpression Trees) is introduced.

NEXT was designed to include the following key properties: a variable-length solution representation with automatic sizing of individuals, an efficient interpretation of solution representations, a diverse repertoire of search operators, and the ability to be customized to work on multiple problem domains, including mathematical modeling via symbolic regression, and neuroevolution (the evolution of artificial neural networks).

The approach was tested using a selection of problems involving symbolic regression of polynomials of different degrees, and neuroevolution for logic synthesis and pairwise classification. Our experimental results, compared against those of Gene Expression Programming on the same problem set, demonstrate that NEXT was capable of successfully evolving variable-length solutions to these problems.

Acknowledgements

Dr. Marcus Vinicius dos Santos for his assistance, guidance, patience, endless reviewing, tolerance of my barrages of e-mails, and an innate ability to come up with acronyms.

My parents Christopher and Jill, without the support and encouragement of whom this work would never have been possible.

My brother Jason for his support, the motivation to return to school, and the occasional well timed kick in the pants.

Dr. Rixi Abrahamsohn for helping me find my way.

Dr. Katrin Rohlf for spending countless office hours explaining things I ought to have learned years before, and for sparking my interest in research.

Lori Christie for being ”just Lori”.

Travis Christie Dunk for adding some colour to my work, and for providing a reminder of the truly important things in life (like apple juice and fast cars).

Dave Gerencer for snow trudging, brewing, Texas, and listening to me talk about chromosomes and fitness functions for two years.

Maria Landau for her endless assistance in all administrative matters, and for the cheerful encouragement.

Michelle Walker for understanding.

Thanks to Cory Baker, Nigel Browne, Tim Dafoe, Joel Micallef, Rajan Parthasarathy, Jesse Robertson, Michael Robertson, Ryan Shaw, Matt Watson and numerous others for their support, encouragement, LaTeXwizardry, CPU time, and distraction.

Additional thanks to Elliott Brood, Sam Calagione, and various purveyors of $C_8H_{10}N_4O_2$-based beverages.

Dedication

This thesis is dedicated to memory of my grandfather Roy Wilson, who taught me the value of learning; whatever form it may take.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The central thesis of this work is that the evolvability of tree-shaped symbolic structures can be improved by an EA that operates on a linear, variable-length encoding of problem solutions that accounts for neutral variation, self-adapts the solution size, and indirectly searches the solution space via a decoding step.

To verify this hypothesis, a novel genetic programming (GP) methodology called NEXT (Next Encoding of eXpression Trees) was developed. Unlike conventional GP approaches which search a space of tree-shaped representations that directly encode solutions to a problem, NEXT belongs to the class of evolutionary algorithms (EAs) called Developmental Genetic Programming (DGP) as it employs indirect encodings to distinguish between the representation of the structures subject to genetic modification (the genotype) and the representation of the structures subject to selection (the phenotype). Indirect encodings have become instrumental for the evolution of highly complex structures, reaching beyond the evolution of computer programs.

The development of NEXT was inspired by a desire to introduce several key characteristics to expression-tree based genetic programming systems:

- **Variable-length solution representation:** Representations relying on fixed length structures require *a priori* knowledge to produce solutions of optimal size. A representation capable of varying the length of the solution during a run will allow for automatic optimization of size.

- **Wide range of genetic operators:** A good representation should provide a flexible organization and allow for a wide range of genetic operations to be performed. Such a

representation should permit operations such as transposition, insertion, and wholesale replacement of elements with ease.

- **Efficiency in chromosome evaluation:** Traversal and evaluation of expression trees can be a complex task, resulting in computationally costly evaluations. Rapid evolution of fit individuals can be aided by a lower complexity evaluation algorithm capable of evaluation in linear time.

- **Flexibility of representation:** A flexible representation should be capable of evolving solutions in a wide range of problem domains. Inclusion of facilities for evolving neural networks in addition to mathematical modeling problems typically demonstrated with evolutionary algorithms is essential.

NEXT solutions permit variation of length during runtime and provide for mixed population sizes. The variability of chromosome length allows for a great deal of genetic operations to be implemented with ease, and allows good solutions to be formed for many types of problems. The organization of NEXT chromosomes also provides for evaluation of solutions in linear time.

1.1 Approach

The characteristics described above served as guiding principles of the design of NEXT, resulting in an indirect encoding centered on the notion of representing sub-expression trees as modular units. Each unit, termed a *cistron*, consists of one root node (a *function*) and an appropriate number of leaf nodes to satisfy the input requirements of the function.

The introduction of the cistron as an entity means that solutions may now be constructed in a modular fashion, and in the evolvable form solutions consist simply of a list of these building blocks. The impact in terms of flexibility of this structure is significant. Rather than directly encoding the expression tree as a whole, we are now able to consider smaller portions of it in isolation.

The genotype-phenotype mapping employed in NEXT does not have any specific requirements regarding solution size, beyond the need for each cistron to contain the appropriate number of arguments for the function. Solutions are permitted to change size during runtime through a variety of genetic operators designed for this purpose. New cistrons may be added anywhere within the solution, and conversely removed at any position. These units may also be transposed in order, transfered or copied between solutions, or replaced.

This results in a highly flexible structure which satisfies the desired characteristics described above. Solutions are automatically sized to the appropriate length during runtime as principles of natural selection promote those solutions which most optimally satisfy the problem. The lack of dependence on predetermined size results in a heterogeneous population which promotes genetic diversity among the evolved solutions.

The organization of NEXT solutions is such that the expression tree need not be constructed for fitness evaluation to take place, and no translation to an alternate form is required, as is the case with other representations. NEXT solutions are evaluated in linear time on the basis that any cistron with no children, ie: leaf nodes of the expression tree, may be evaluated in isolation. The evaluation results of a given sub-expression tree are simply passed to its parent, resulting in a marked decrease in the computational complexity of fitness evaluation.

The usefulness of an evolutionary algorithm is determined not only by its ability to evolve good solutions, but also by the breadth of problems which it is capable of representing. With this in mind, NEXT was designed to be capable of evolving both mathematical modeling solutions as well as neural networks. The capability for induction of neural networks including weights and thresholds, as well as topology greatly increases the scope of the problems NEXT can tackle.

1.2 Contributions

The principal contribution of this thesis to the body of work in the field of evolutionary computation is the development of NEXT, a new representation and evolutionary algorithm. The

components of this include:

1. The introduction of the cistron, an entity designed to represent a sub-expression tree consisting of one function and *n* leaf nodes in a self-contained unit capable of being manipulated in ways not possible in Gene Expression Programming.

2. Design of a method for evaluating NEXT chromosomes in linear time, representing a significant performance increase over existing GEP-based methods.

3. Development of an evolutionary algorithm accompanying the new representation, providing operators for the manipulation of cistrons and other entities of NEXT.

1.3 Overview of Thesis

The remainder of this thesis consists of four chapters and an appendix: Chapter 2 - Background and Related Works, Chapter 3 - Methodology, Chapter 4 - Results, Chapter 5 - Conclusions and Future Work, Appendix A - Glossary.

Chapter 2 presents the background material necessary for an understanding of the work presented here. Discussed are concepts of evolutionary computation, an overview of the evolutionary algorithms which form the lineage of NEXT, and concepts relating to neuroevolution.

Chapter 3 discusses the methodology used in this dissertation, including a detailed overview of NEXT, the demonstration problems used in validating this work, and the methodology by which the testing was conducted.

Chapter 4 outlines the results obtained during testing and provides an analysis of these results relating to the central thesis of this work.

Chapter 5 presents some conclusions on this work, and outlines potential areas for future enhancement of NEXT.

Appendix A provides a glossary of the terminology used in this work. Jargon highlighted with *emphasis* in the chapter text can be found defined in this appendix.

Chapter 2

Background and Related Work

This chapter presents the background information necessary for the understanding of the work presented in this thesis. Discussed herein are elementary concepts of evolutionary computation, prior methodologies which can be considered the heritage of NEXT, and concepts of neuroevolution.

2.1 Introduction to Evolutionary Computation

This section introduces the fundamentals of evolutionary computation (EC). Its objective is to provide the reader with an understanding of the issues at hand prior to discussion of specific EC methodologies. As a great amount of the terminology used in EC can take on slightly different meanings depending on the author, this section also seeks to clarify and define the major terminology as it is understood and used in this work.

2.1.1 Basis of Evolutionary Computation

Evolutionary computation is centered around the notion of applying concepts of natural selection to the solution of problems or the induction of computer programs. The algorithm described in Algorithm 1 forms the basis of all EC methodologies, with variations to suit the particular implementation. Specific differences from the algorithm can include the method of population creation, the selection mechanism used, and the methods of reproduction and modification.

In Algorithm 1, and through this work, an *individual* is a potential solution to the user-defined problem the EC system is set to solve. A population is a collection of individuals upon

Algorithm 1: Basic Evolutionary Computation Algorithm

```
begin
    create initial population
    evaluate fitness of individuals
    while termination criteria not met do
        selection
        reproduction and/or modification
        evaluate fitness
    end
end
```

which search operators such as selection, reproduction, and modification will act.

2.1.2 Representation, Encoding, and the Genotype-Phenotype Duality

Two concepts which are as central to this work as they are inextricably linked are those of *representation* and *encoding*. Representation can be viewed as the definition by a particular methodology of what possible solutions to a problem look like [3]. Encoding is looked at in this work as the EC-system-understandable rendering of a solution representation.

Many different encodings and representations are used in EC methods including those based on binary strings, boolean values, graphs, and trees. [3]

The concepts of *genotype* and *phenotype* are closely related to those of representation and encoding. Genotype can be broadly defined as the entity upon which genetic operators act. In biological terms, the DNA of an organism can be thought of as the genotype. The term phenotype refers to the entity upon which selection acts, which in a biological sense refers to the body of an organism interacting with its environment.

In order to illustrate concepts of EC in a methodology-neutral fashion, below we will introduce a fictitious genetic representation consisting of simple geometric shapes. The language of the encoding uses the letters `c`, `t`, `s` to represent a geometric circle, triangle, and square respectively.

Figure 2.1 illustrates a sample encoding in this language for an individual consisting of four positions. The genotype of this individual is a simple linear arrangement of characters from the

language of this EC system.

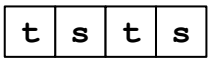

Figure 2.1: A simple genotype

Figure 2.2 illustrates the phenotype corresponding to the encoding of the genotype shown in Figure 2.1. In this encoding, the symbols in the genotype are translated from characters in a linear string to a phenotype consisting of a series of geometric shapes with properties.

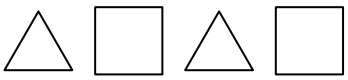

Figure 2.2: Phenotype for the genotype shown in Figure 2.1

The relation of genotype and phenotype to representation and encoding varies by the EC method under discussion. Two types of encodings are prevalent in EC works, *direct encoding* and *indirect encoding*. A direct encoding method considers the genotype and phenotype to be one and the same, meaning both genetic operators and selection operate upon the same entity. An indirect encoding considers the genotype and phenotype to be separate entities and utilizes a *genotype-phenotype mapping* (GPM) to translate between the two. The GPM is an algorithm which translates the encoded genotype and results in the expressed phenotype. EC systems employing a GPM will typically use the genetic operators to modify the genotype, and perform fitness testing and selection based upon the effectiveness of the expressed phenotype. [11]

Banzhaf [2] discusses the notion that in a system employing a GPM two separate spaces exist, the *search space* of the genotype and the *solution space* of the phenotype. In such a system the search mechanism operates solely on the genotype as the genetic operators make modifications to the encoded individuals. The space of all possible combinations of the symbols of the encoding can be considered the search space. The solution space can be thought of as the space consisting of all possible phenotypic expressions. To examine this concept, we will limit the language of our fictitious system to only two symbols: **`t`** and **`s`**. As well, we will limit the length of possible encodings to two positions in order to minimize the number of potential solutions so that the entire search and solution space can be displayed compactly. The left

column of Figure 2.3 contains all possible combinations of the genotypic symbols, and thus represents the entire search space of this representation. The right column contains all possible combinations of the phenotypic symbols, and accordingly represents the entire solution space.

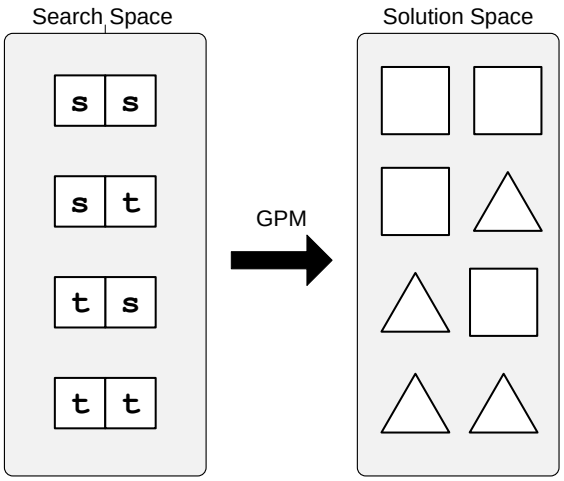

Figure 2.3: An example of search space and solution space

It is easy to see the entire search space when there are a limited number of symbols in the language and a limited individual size. In this instance, relationship between the number of variables and the size of the individual is only 2^2 in dimension. If we consider expanding the individual to contain three symbols, the search space expands again accordingly. The same principle of space expansion holds true with an increase in the number of symbols in the language of the encoding.

The argument for the use of a GPM consists of several points: firstly, it is much easier to represent an encoded solution and operate upon its constituents than it is to make modifications to the phenotype. Secondly, the use of a GPM allows for greater variation in the composition of solutions. Kimura [18] discusses the concept of neutrality in biological evolution, stating that diversity in natural biology is mainly due to variations in the genotype which have little to no effect on the phenotype. Banzhaf [2] continues this line of thought by introducing the concept of *neutral variation* to EC.

The use of a GPM will allow operations within the search space to manipulate the genotype without constraint, instead imposing restrictions on the expression of those individuals as phenotypes. Consider a genotype-phenotype mapping for the geometric shape language presented

earlier in this chapter which limits the maximum number of geometric sides in a phenotype to be no greater than 11, yet places no constraints on the genotype. Figure 2.4 shows three distinct genotypes, which when translated through the GPM, result in an identical phenotype. The GPM in place simply ignores the fourth position as the first three positions of square, triangle, square contain 11 sides each and meet the requirement.

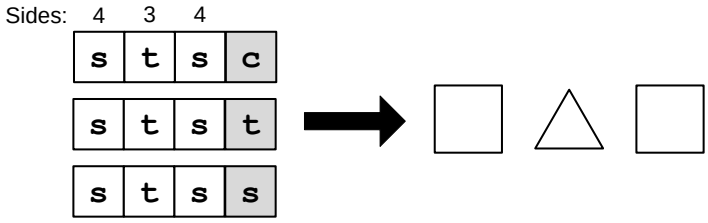

Figure 2.4: Different genotypes resulting in identical phenotype

This relationship of many genotypes to one phenotype may seem redundant and wasteful, but its usefulness is demonstrated by applying a simple mutation to the three genotypes of the above example. As seen in Figure 2.5, the mutation of the third position to a circle from a square changes the total number of sides of the first three positions to 8. This allows for the fourth position to be expressed for two of the phenotypes, resulting in three distinct phenotypic expressions from the three genotypes.

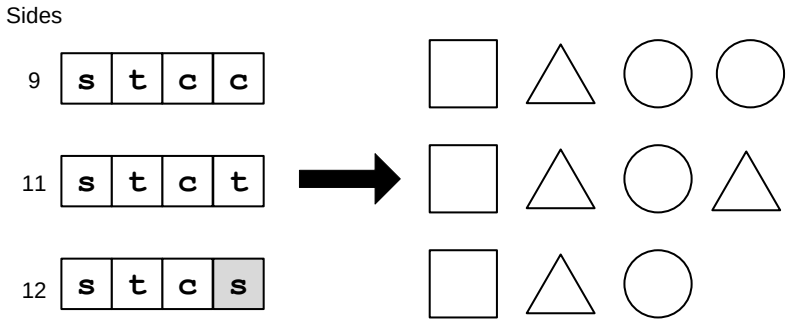

Figure 2.5: A small mutation results in three distinct phenotypes

A second noteworthy effect of many-to-one mappings is the possibility of a mutation of the genotype having no effect on the phenotype. The possibility exists using such a mapping that a mutation may take place in a region of the genotype which is not expressed at this time, resulting in a modification which is neutral in regards to the phenotype.

2.1.3 Genetic Operators

The modification of individuals within a population is accomplished through the use of genetic operators. These operators provide a means through which genetic diversity is introduced and maintained within a population. There has been a growth in the number of genetic operators implemented in subsequent EC systems as the field has developed, and the genotypes have become more complex. More specific operators allow for more refined modifications to individuals within the population and a wider array of modifications to be made.

The majority of genetic operators used in EC systems take inspiration from the biological operators of recombination, mutation, and straightforward reproduction [3]. Further operators introduced, such as those in NEXT, are variations on these themes which provide a more granular approach to the modifications while the basic idea remains the same. *Recombination*, also termed *crossover* is an operation which combines the genetic material of two parents by swapping portions of the genotype of both parents to create new offspring. *Mutation* is defined for EC purposes as the random change of one or more positions of a genotype to a randomly selected symbol. Reproduction consists simply of the strict copying of genetic material from one individual to another. These operators are illustrated in the following figures using simple geometric shapes as the genotypic material to illustrate the concepts in a representation-neutral manner.

Figure 2.6 contains a simple illustration of the concept of recombination. Two parents are shown consisting of four shapes each, Parent 1 consists entirely of triangles and Parent 2 consists entirely of squares. Through the operation of recombination two offspring are produced, with recombination occurring in the region shaded in grey. It can be seen that the elements in that region were exchanged between the two parents and the new offspring contain a mixture of the genetic traits of both parents.

The concept of mutation is illustrated in Figure 2.7. The genotype of the individual shown at top consists of four triangles prior to undergoing mutation. The mutation illustrated in the second genotype takes place at the fourth position where a triangle is mutated randomly to a

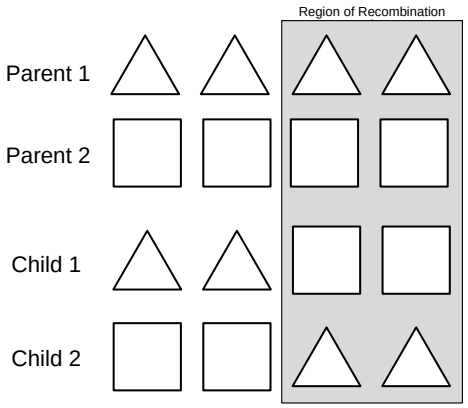

Figure 2.6: Recombination

square.

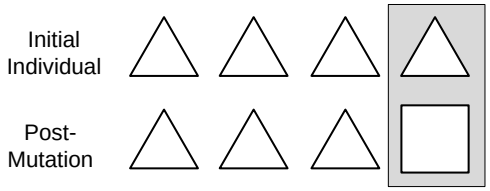

Figure 2.7: Mutation

Replication is the simplest operator in use in EC systems, consisting of the faithful copying of genetic material from one individual to another, where the genotype remains the same and is merely inserted into the new population.

2.2 Evolutionary Algorithms

This section discusses the evolutionary algorithms (EA) which can be considered the lineage of NEXT. The earliest genetic algorithms presented by Holland [14] and DeJong [4] proposed the notion of evolving solutions through the mating of fixed length binary strings. Koza [19] refined the concept through the use of a tree-based representation with the introduction of Genetic Programming. The use of expression trees in an evolutionary approach was further expanded upon by Ferreira [7] with the genotype-phenotype mapping of GEP.

The basis of all EA presented here is the process flow found in Figure 2.8. From the initial attempts to emulate natural selection in software, to current methods, all methods rely on

the basic principles of an initial population being modified through various means until target criteria are met.

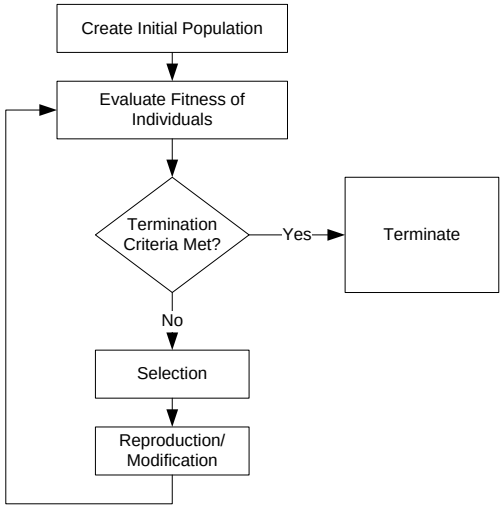

Figure 2.8: Basic evolutionary algorithm

2.2.1 Genetic Algorithm

The methodologies employed in the original Genetic Algorithm (GA) works [14], [4] have formed the basis of much of EC since their introduction. The ideas presented in Figure 2.8 emerged from these early works and continue to be implemented in more modern approaches.

The genotypic structure of the early GA involved a linear arrangement of binary strings, considered a universal genetic code by Holland [14]. The emphasis of this generation of EA was on two things: allowing many problems to be represented by the same style of binary genotypes, and a focus on evolving a population of N individuals to replace the prior generation of N individuals through genetic modification. [5].

The genetic operators implemented in such systems consisted of mutation through simple changes of individual bits within the binary strings, and one-point recombination where a given position was exchanged between two individuals.

Of note is the early GPM employed by GAs in order to represent diverse problems in a simple binary format. As the genotype consisted only of binary strings, a mapping was required to translate the evolved solution back into the problem domain.

2.2.2 Genetic Programming

Genetic Programming (GP) has been one of the most influential approaches in EC since its introduction by Koza in 1992 [19]. Rather than the linear genotype of GA, GP is based upon the use of trees in representing individuals. The primary aim of GP was to develop a mechanism for evolving computer programs, as Koza felt that conventional structures used in machine learning were nothing like computer programs [19]. GP was developed on the basis of two central points: firstly, a wide variety of problems can be restructured to be problems of computer program induction, and secondly that GP provides a mechanism for doing so.

The structure of GP represents an increase in complexity over previous EA and allows for varying size and shape of individuals, in contrast to the fixed-length linear structures of earlier works. As the initial implementation of GP was developed in LISP, the terminology of GP is somewhat LISP-centric, and we will refer to entities within GP structures by more implementation-neutral terms which are applicable to other tree-based methodologies as well.

The organization of GP trees is based upon the notions of *functions* and *terminals*, borrowing the concepts from ordinary programming languages. A function is analogous to a function in programming, where some task is performed. Typical GP functions include arithmetic operators, logical functions, and programming operators. Each function has an associated *arity*, which is the number of arguments that function requires. For instance, functions of addition or subtraction require two arguments and thus have an arity of two. Functions reside at the root and internal nodes of the tree, and their children may be either functions or terminals depending on the construction of the tree. Terminals represent the input to the program and only exist at the leaf node level.

Figure 2.9 [19] shows an even-2-parity function constructed of the following function and terminal sets: **`F = {AND, OR, NOT}, T = {D0,D1}`** where **`D0`** and **`D1`** are inputs to the logical function.

The canonical implementation of GP primarily uses replication and crossover as a means of evolving the population. The replication scheme used is a straightforward copying of individ-

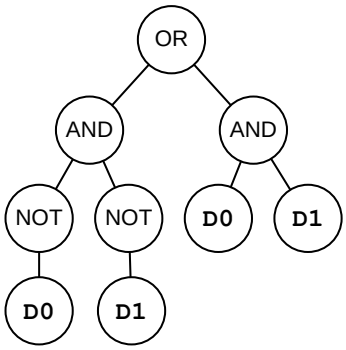

Figure 2.9: Sample GP individual

uals selected according to fitness into the new population of the next generation. The variation in GP populations comes through the use of a subtree-swapping crossover operator.

Figure 2.10 illustrates a simple GP individual using the function and terminal sets defined previously. It can be seen that in Individual A, the internal nodes consist solely of the **AND** function, and in Individual B solely of the **NOT** function. The regions of the tree shaded in grey have been randomly selected as the subtrees which will be swapped in crossover.

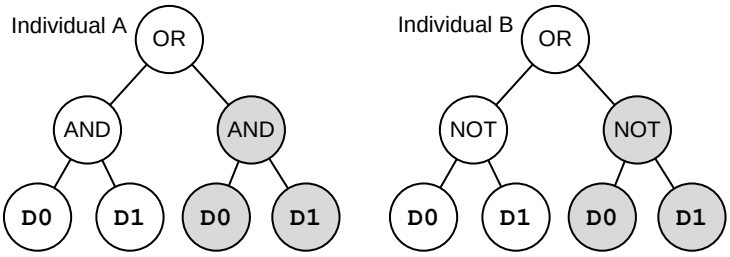

Figure 2.10: Before GP crossover

Figure 2.11 shows the state of the offspring of the original two individuals after the crossover operation. Note that the rightmost subtree on both individuals have been exchanged, sharing the genetic material of both parents.

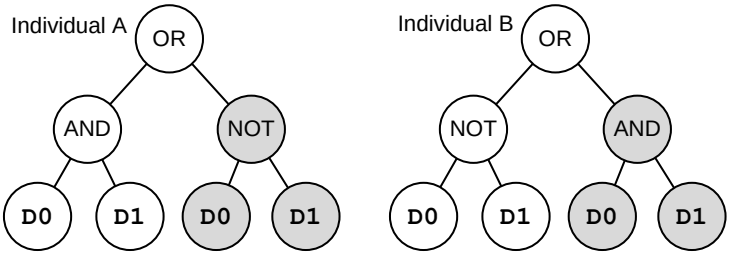

Figure 2.11: After GP crossover

Unlike systems employing a method of GPM, GP makes no distinction between genotype

and phenotype. Each tree represents an individual, and operations are performed by acting directly upon the tree. In the case of an implementation in LISP, sections of the list are swapped between individuals during crossover.

The notion of *bloat*, that is the excessive growth of individual size, has been discussed [12] as an issue with GP. There can be a tendency of GP programs to increase in size well beyond the optimal dimensions for a particular problem. A common method of controlling bloat within GP individuals is the application of *parsimony pressure* [23]. Parsimony pressure refers a method of controlling individual size by rewarding individuals of smaller dimensions. Individuals possessing a size determined to be optimal are said to be parsimonious.

2.2.3 Gene Expression Programming

Developmental genetic programming (DGP) [17] was initially introduced as an EC methodology which employed binary strings as genotypes [3], and a mapping to complex structures denoting the phenotype. [2]. In more recent times, the term DGP has grown to encompass an entire class of methods which espouse this concept in various differing methodologies.

Gene Expression Programming (GEP) is a DGP methodology proposed by Ferreira [7] and first published in 2001. As with other genetic programming methodologies, it follows the basic process of creating a random population of individuals and proceeding to narrow the search for an ideal solution through means of modification, reproduction, and selection.

Where GEP differs from the approaches described above is in the representation and encoding used for its individuals. Unlike GP which made no distinction between the genotype and phenotype, GEP encodes the individual as a fixed-length linear string and employs a genotype-phenotype mapping to translate the linear string to the expression tree of the phenotype. The elegance of the GPM used by GEP is in the potential for unexpressed terminals, which allow for neutral mutations to accumulate as evolution progresses. As these changes collect throughout evolution, mutations elsewhere in the gene may trigger their expression and shift the genetic search to a new portion of the search space.

A GEP gene is composed of two distinct regions which enforce the structural organization of the expression tree. The first region, the head consists of both functions and terminals and the latter region or tail consists of only terminals. The size of a gene in GEP is governed by Equation 2.1, where t is the size of the tail, h denotes the size of the head, and n is the maximum arity of the function set.

$$t = h(n-1)+1 \tag{2.1}$$

All GEP genes are encoded in a notation called the Karva language, which uses simple characters to represent functions and terminals. Inputs to GEP genes are represented by lower case letters beginning with `a` and continuing through the alphabet as required.

To facilitate the understanding of the structural relationships in GEP, consider the mathematical function of Equation 2.2 and its accompanying genotype (Figure 2.12) and phenotype (Figure 2.13) [9]. Equation 2.2 consists of a simple square root of a product, and is representable compactly in GEP.

$$\sqrt{(a+b)*(c-d)} \tag{2.2}$$

Figure 2.12 illustrates the encoding of this equation as a linear genotype in Karva notation. The function set used consists of **F=**{**Q,+,-,***} and the terminals of **T=**{**a,b,c,d**}. Note that `Q` is the Karva representation for square root. Also note that this example does not illustrate a complete GEP gene, but rather omits unexpressed positions for simplicity.

Q	*	+	-	a	b	c	d

Figure 2.12: GEP gene for Equation 2.2

The example illustrated above represents what is termed in GEP parlance to be an *unigenic chromosome*, that is a chromosome consisting of a single gene. GEP also includes support for *multigenic chromosomes* whereby a chromosome consists of multiple genes within the single entity. Figure 2.14 illustrates such a chromosome, consisting of three individual genes. Multi-

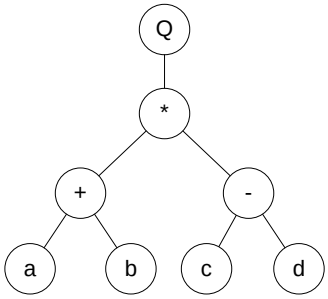

Figure 2.13: Expression tree for Equation 2.2

genic chromosomes such as this 3-genic example are useful in a variety of situations where one may wish to decompose a problem into separate parts or evolve solutions to distinct problems within one chromosome.

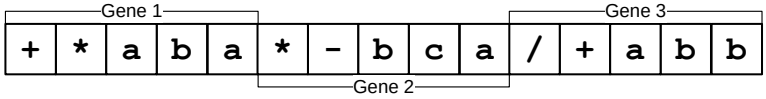

Figure 2.14: A 3-genic GEP chromosome

The example in Figure 2.12 does not contain the unexpressed positions as they were omitted for clarity. By Equation 2.1, the gene shown above should have one additional position in the tail. These unexpressed positions are termed the *non-coding region* in the terminology of GEP, and contribute to the genetic diversity of the individual. The additional position would be required if the `Q` function mutated to a function with an arity of two. The contribution of the non-coding regions of GEP chromosomes is twofold. Firstly, it allows for expression trees of various sizes to be expressed by a fixed-length genotype through the use or non-use of all of the available terminals. Secondly, as discussed, the concept of neutral variation enables modifications in the non-coding regions to be stored until such a time when a genetic change expresses them, shifting the search to a new portion of the search space.

GEP implements a greater variety of genetic operators than either GA or GP, owing to both the increased complexity of the representation and an interest in increasing the methods by which genetic variation can be introduced. The genetic operators of GEP are implemented according to probabilities set prior to execution of the system. Individual chromosomes are selected according to the probabilities for each operation, *ie:* mutation rate, recombination rate,

etc., and the operators are then applied before the resulting individuals are inserted into the new population. In order to maintain the structural organization of the genes, and ensure that the genotype-phenotype mapping is functional, all genetic operators within GEP must refrain from placing functions within the tail of the gene.

Operators implemented in the canonical version of GEP include: replication, mutation, recombination, and transposition. Replication is simply the copying of a selected chromosome into the new population without any genetic change. Mutation in GEP is a simple point mutation which changes one randomly selected position to a new random function or terminal as appropriate for the region.

Recombination (or crossover) is implemented much as in other genetic programming systems, where genetic material is exchanged between two parents. Two variants are implemented, one-point recombination and two-point recombination. In the former, two genes of equal length are aligned and one point is selected as the crossover point. The material downstream of the crossover point is exchanged between the two parents to form two new offspring. Two-point recombination operates similarly, except in addition to a crossover start point, a crossover end point is also selected and the material between the two points is exchanged. Figure 2.15 illustrates recombination, with the crossover point randomly selected to be between the third and fourth positions of the gene. The right column of the figure shows the result of the operation, with the genetic material downstream of that point swapped in the two offspring.

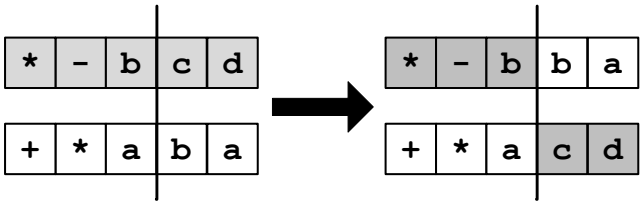

Figure 2.15: Recombination in GEP

A concept incorporated into GEP not present in GA or GP is that of transposable elements, termed IS elements and RIS elements depending on the context. IS elements, or insertion sequence elements, are a fragment of the gene consisting of functions or terminals which transpose to the head of the gene. RIS elements (root insertion sequence elements) follow the same

concept, but must start with a function as they transpose to the first position of the gene. The length of both IS and RIS elements is defined by the user prior to runtime of the system, and may be selected randomly from a list of several lengths.

Our work [16] with GEP (specifically in the GEP-NN neuroevolution variant) brought to light several issues with GEP, which became the greatest influence on the development of NEXT. These concerns include: the destructive nature of some GEP genetic operators, the limitations of the fixed-length individual, and the computational cost of evaluating GEP chromosomes. The destructive nature of GEP genetic operators is more evident in individuals with functions of higher arities, such as those used for neuroevolution. As such, it will be discussed further in Section 2.3.1

Although beneficial to evolution, the GPM used in GEP requires a great deal of computational effort when evaluating the fitness of individuals. The suggested method for evaluation is to transform the linear genotype into a matrix such as shown in Figure 2.16. The matrix must then be traversed row by column and the terminals replaced with their values, and the result of each function calculated.

	0	1	2	3
0	Q			
1	*			
2	+	-		
3	a	b	c	d

Figure 2.16: Matrix representation for the expression tree for Equation 2.2

This method, at a cost of O(n^2), (where n is the size of the individual) becomes especially expensive as the size of the head increases. With this the size of the matrix increases proportionally, and thus the cost by an order of magnitude. NEXT was designed to avoid this type of requirement and to lower the computational cost of fitness evaluation.

2.3 Neuroevolution

Neuroevolution is the sub-field of evolutionary computation which deals with the induction of artificial neural networks through evolutionary means. Early neuroevolution approaches were designed to either evolve the weights of neural networks or the topology as opposed to complete induction of the network. A newer class of systems called Topology and Weight Evolving Artificial Neural Networks (*TWEANN*) has emerged which is capable of full neural network design.

2.3.1 Neuroevolution in GEP

Neuroevolution is possible in GEP through a series of extensions to the original GEP algorithm, called GEP-NN [8]. The modifications retain the structure of the GEP representation such as the head and tail regions, or domains, but introduce new domains to the chromosome to make possible the complete induction of neural networks including weight, thresholds and topology.

As is often the case with genetic programing applications, a suitable phenotypic representation must be developed to allow the problem to be evolved under the constraints of the evolutionary system. Artificial neural networks may have multiple parent nodes connecting to one child, such as in the sample neural network shown in Figure 2.17. This arrangement is not possible in the tree-based phenotype of GEP, and an alternative representation had to be found. Figure 2.18 illustrates a GEP-suitable arrangement of the same neural network, with multiple copies of each input used to represent the cross links (labeled 2 and 3 in Figure 2.17).

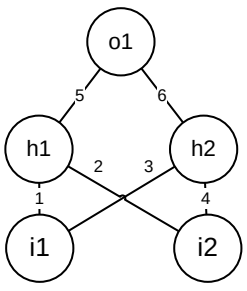

Figure 2.17: Conventional neural network

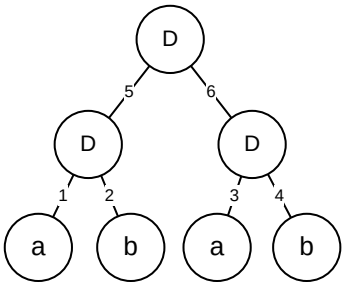

Figure 2.18: GEP-NN representation of Figure 2.17

This phenotypic representation required the development of a function set to represent the topology of neural networks. The function set of GEP-NN consists of four functions which represent the arity of the nodes within the neural network. Functions of `U,D,T,Q` represent arities of one, two, three, and four respectively. The effect of neutral variation becomes greater in GEP-NN problems, as the functions of `T` and `Q` possess higher arities than those typically used in mathematical modeling, *ie*: addition, subtraction, *etc*. As the length of the tail is dependent on the maximum arity of a function, a GEP-NN gene may contain more introns than is likely in a problem with a maximum arity of two. Figure 2.19 illustrates this effect with two genes, the first a GEP gene with a head size of two, and a function arity of two. The second gene is a GEP-NN gene, also with a head size of two and two functions of arity two as well. The higher maximum arity (four) of GEP-NN functions produces a longer tail, and thus the four codons shown shaded in grey are not expressed in this example.

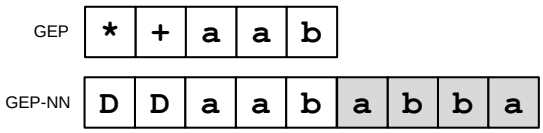

Figure 2.19: Comparison of non-coding regions in GEP vs GEP-NN

The representation of a neural network also required additional information not needed for the evolution of the mathematical modeling and logic synthesis problems GEP is customarily used for. In order to achieve full induction of a neural network, a means to evolve the weights and thresholds must be included. GEP-NN chromosomes contain two additional domains, which store array indices of floating point valued weights and thresholds. The weights and thresholds are stored in two separate arrays, typically of ten elements each.

Figure 2.20 shows an example GEP-NN gene with the separate domains of the gene marked for clarity. The weight domain consists of one array index for each edge of the tree. The threshold domain holds one array index for each element of the head, to be used if the corresponding element of the head is a function.

The indices stored within the chromosome correspond to positions within two separate arrays of weights and thresholds stored for each gene within the population, as illustrated within

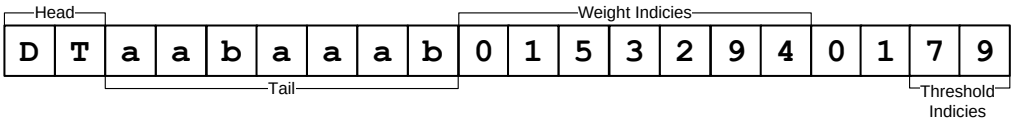

Figure 2.20: Sample GEP-NN gene

Figure 2.21. The values of the weights and thresholds are up to the discretion of the user, but are typically floating point numbers on the range of [-2,2].

```
W = {-1.64, -1.834, -0.295, 1.205, -0.807, 0.856, 1.702, -1.026, -0.417, -1.061}
T = {-1.14, 1.177, -1.179, -0.74, 0.393, 1.135, -0.625, 1.643, -0.029, -1.639}
```

Figure 2.21: Associated weight and threshold arrays

GEP-NN also includes additional genetic operators for the evolution of the weights and threshold indices and values. The GEP-NN operators for transposition, recombination, and mutation act upon both of these regions, and provide a mechanism through which weights may be moved to different edges within the network, or changed entirely.

Figure 2.22 illustrates the phenotype resulting from the genotype-phenotype mapping of Figure 2.20. Note that the functions of **D** and **T** connect to two and three child nodes respectively.

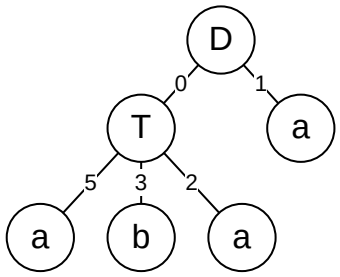

Figure 2.22: Phenotypic expression of Figure 2.20

GEP-NN was demonstrated by Ferreira in both unigenic and multigenic form on problems of logic synthesis through neural networks. Solutions evolved for XOR and a 6-Plex Multiplexer were demonstrated [8] to evolve compact solutions for both problems.

Our work [16] with GEP-NN focused on the evolution of neural classifiers for several data sets, and was successful at demonstrating that neural networks for problems of a higher complexity can be evolved with GEP-NN. However, in addition to the issues discussed in Section

2.2.3, some more GEP-NN specific problems were discovered which resulted in less than optimal results from the GEP-NN-evolved neural classifiers.

The issue of fixed-length genotypes is of particular concern when attempting complex problems such as the neural classifiers discussed in [16]. A problem constantly encountered in attempts to evolve classifiers was that of determining an optimal chromosome size prior to evolution. This resulted in many repeated runs of the system with incremental growth in chromosome size in an attempt to find the correct balance between large enough chromosomes to include all the required terminals, and small enough chromosomes to evaluate quickly.

A second issue which plagues complex GEP-NN chromosomes more so than those of less complex mathematical modeling problems is the issue of destructive genetic operators. The nature of the genotype-phenotype mapping in GEP means that a shift in the arity of a function can severely disrupt relationships between functions in terminals in subtrees other than the specific one being mutated.

Especially in neuroevolution problems, the relationship between a function and given terminals can be essential to network function and should not be disrupted by changes elsewhere in the gene. The GPM employed in GEP shifts terminals from one function to another as the arity changes, as is illustrated in the following simple example. Figure 2.23 illustrates a sample point mutation at the second position from an arity of two to an arity of four.

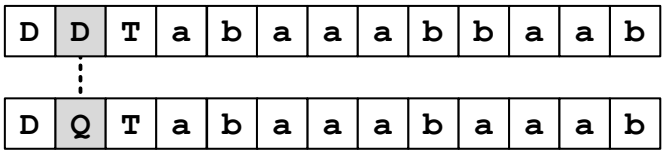

Figure 2.23: A sample destructive point mutation on a GEP-NN chromosome

Figure 2.24 shows the phenotype corresponding to the sample gene in Figure 2.23 prior to the mutation operation. The lower-rightmost subtree consists of the function **T** and three terminals, **a,a,a**. As can be seen in Figure 2.25, the mutation of the second function to a higher arity shifts the terminals. When four terminals are assigned to the new function **Q** at the second position, two terminals are taken which were previously assigned to the subtree to the right, resulting in the relationship between **T** and **a,a,a** being broken, and the terminals

a,b,b residing in their place.

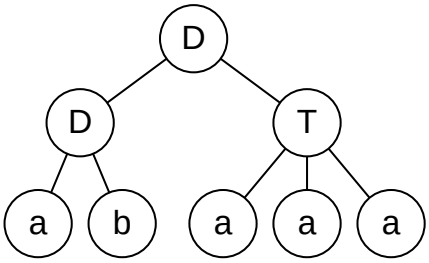

Figure 2.24: Phenotype of Figure 2.23 prior to mutation

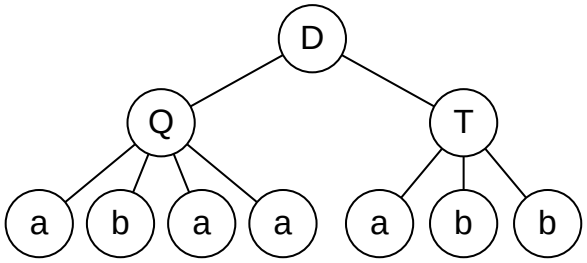

Figure 2.25: Phenotype of Figure 2.23 after mutation

Going on the assumption that **Taaa** could be a valuable relationship to the solution of the problem at hand, a modification to an unrelated subtree has destroyed the relationship. The representation of NEXT has been designed specifically to avoid this problem and protect individual subtree relationships.

2.3.2 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) is a modern method of neuroevolution introduced by Stanley [24]. NEAT belongs to the class of algorithms called TWEANNs, or Topology and Weight Evolving Artificial Neural Networks. It is a complete methodology for the efficient evolution of neural networks.

NEAT introduces a concept called *complexification* whereby individuals in the population initially are of a small size, and grow through evolution to a more appropriate dimension for the problem to be solved. Stanley cites biological evidence [22] that natural evolution uses such a process by adding more genes to add complexity to the phenotype.

The motivation for such a methodology in EC is to avoid searching an unnecessarily large portion of the search space. The small size of the initial members of the population means the area being searched is initially small, and grows in complexity with the size of the individuals. This allows for a more efficient search of the space than starting with an arbitrarily large solution, as the solution may be found in the lower regions of the space.

Chapter 3

Methodology and Implementation

As stated in Chapter 1, the hypothesis we put forward here is that the evolvability of tree-shaped symbolic structures can be improved by an EA that operates on a linear, variable-length encoding of problem solutions that accounts for neutral variation, self-adapts the solution size, and indirectly searches the solution space via a decoding step.

To verify this hypothesis, a novel developmental genetic programming methodology called NEXT (Next Encoding of eXpression Trees) was developed.

This chapter provides an introduction to the concepts of NEXT demonstrated in this work. The entities of NEXT are first discussed to give the reader an understanding of the structure and organization of the solutions produced by NEXT. The chapter then introduces the evolutionary algorithm and genetic operators of NEXT, and discusses how NEXT addresses the principles which guided the development of NEXT, namely:

- Variable-length solution representation
- A wide range of genetic operators
- Efficiency in chromosome evaluation
- A representation capable of evolving solutions for a wide range of problem domains.

Lastly, the methodology used in evaluating the effectiveness of NEXT is introduced and the demonstration problems discussed.

3.1 NEXT

NEXT is a developmental genetic programming methodology which seeks to build on the lineage of GP and GEP, while introducing new concepts in the structural organization and encoding of problem solutions, and genotype-phenotype mapping to alleviate the issues with prior methodologies discussed in Chapter 2.

NEXT permits a variable-length representation to allow for the discovery of the optimal gene size for a given problem. Genetic operators are included to modify the size of individuals, and allow for the recombination of disparately sized individuals.

The encoding of NEXT is organized around the concept of the tree-shaped structures, and protects relationships within these subtrees from the effects of genetic changes within other subtrees.

The genotype-phenotype mapping employed in NEXT is of lower computational complexity than that employed by other methods, such as GEP. The GPM of NEXT is capable of representing an expression tree as a linear structure with known parent-child relationships, and bypasses the need to build costly two-dimensional representations of the phenotype in order to evaluate a given individual.

3.1.1 The Entities of NEXT

Figure 3.1 illustrates the hierarchical structure of the entities of NEXT's representation, and defines the relationships between them. In the figure, n denotes the number of genes in a chromosome, m denotes the number of cistrons in a gene, and c denotes number of codons in a cistron.

This subsection discusses the entities of NEXT in detail, and outlines their purpose and structure.

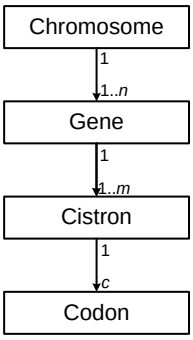

Figure 3.1: NEXT entities

Codon

The *codon* is the lowest level entity within the hierarchy of NEXT, encoding the functions and terminals of the expression tree. In NEXT, a function is an internal node of the expression tree and terminals represent leaf nodes. The meaning of these codons varies depending on the type of problem being solved.

While NEXT is capable of using any function or terminal set, those used in this work have a basis in the Karva language of GEP; one notation for mathematical modeling problems, and one for neuroevolution.

Symbolic regression problems utilize the common arithmetic operator set used in GEP, with functions of F_{Math}={`Q,+,-,*,/`} representing square root, addition, subtraction, multiplication, and division respectively.

Neuroevolution problems use the same function set as that found in GEP-NN: F_{Neural}={`U,D,T,Q`}, for neurons of arity one, two, three, and four respectively.

The terminal set used across all problems is that of lowercase letters, representing the independent variable in mathematical modeling problems, and neural network inputs for neuroevolution problems.

Cistron

In biological terms, a cistron is defined as the locus responsible for generating a protein. This definition is analogous to its meaning in NEXT where the cistron is the lowest level entity which encodes a complete functional structure. Each NEXT cistron consists of *n+1* codons:

one function and *n* terminals, where *n* corresponds to the maximum arity of the function. This structure represents a complete expression tree with one root node and *n* leaf nodes.

The motivation for introducing an entity at the level of the cistron stems from the desire to have a flexible, modular representation with self-contained blocks which can be inserted, deleted, and transposed. The use of cistrons allows genes to grow and shrink without resizing any elements, rather new cistrons are simply inserted or deleted from the gene. This is in contrast to methods of resizing genes in other representations such as in GEP where substantial changes must be made to the gene.

Figure 3.2 illustrates a neuroevolution cistron consisting of the neural function with an arity of four and the appropriate terminals. Note that in this example, the weights and thresholds are omitted for clarity. Figure 3.3 illustrates a common cistron for the expression **a*a**, "a multiplied by a". Note also that this cistron is smaller than the one shown in Figure 3.2, as the maximum arity of the operators used in symbolic regression problems is two.

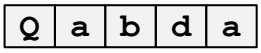

Figure 3.2: Sample neuroevolution cistron

Figure 3.3: Sample symbolic regression cistron

The cistron is also the level at which the neuroevolution elements are present within the chromosome. Each cistron contains weights and thresholds, and is in effect a small neural network in and of itself. Floating point values are stored within the cistron as one threshold and *n* weights corresponding to the function and *n* terminals. Storing the values within the cistron rather than in indexed arrays as is done with GEP-NN allows for substructures to be retained during genetic operations.

Figure 3.1.1 illustrates a complete neuroevolution example with the weights and thresholds. Unlike the array-based collections of weights and thresholds in GEP, NEXT implements *ephemeral constants* for weights and thresholds with genetic operators to modify the values during runtime. Ephemeral constants refers to a method by which one value is stored for a given requirement, and directly changed by modifying operators in lieu of a method of maintaining a bank of constants for use in the system.

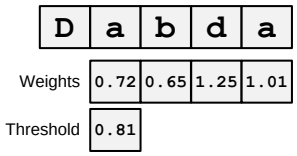

Figure 3.4: Sample neuroevolution cistron with weights & thresholds

A given cistron may have terminals which are unexpressed if the arity of the function is lower than the maximum arity, as would be the case in a neuroevolution example where the maximum arity is typically four and a given function arity may be two or three. This results in one or more unexpressed terminals, which are however necessary to support changes within the cistron such as a mutation of the function to one of a higher arity. Figure 3.1.1 depicts a cistron with a function of arity 2, where the maximum arity of the problem is 4. The last two functions of this cistron would not be expressed during evaluation. Further discussion of unexpressed terminals can be found in Section 3.1.3.

Gene

While the genetic material in NEXT is contained entirely within the cistrons, the gene can be thought of as a container for a collection of cistrons. Each gene consists of one more more cistrons, and can either represent the complete genome of an individual (a candidate problem solution), or part of a larger individual depending on whether a unigenic or multigenic solution is desired.

Figure 3.1.1 illustrates a sample gene with three cistrons. This configuration is intended for illustration purposes only, and during runtime a gene may contain any number of cistrons greater than or equal to one.

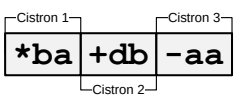

Figure 3.5: Sample symbolic regression gene with 3 cistrons

Chromosome

Chromosomes are entities which consist of one more more genes depending on the desired configuration for the representation of the problem to be solved. For many simple problems, a unigenic solution representation is preferable so as to reduce the size of the individual. Figure 3.1.1 can be considered to represent a unigenic chromosome, as it is a complete gene.

Multigenic chromosomes are those which consist of more than one gene. This configuration presents advantages in certain problems, such as in the classification work previously undertaken by the authors in which a given gene represented a classifier for one specific class of a multiclass problem [16].

Figure 3.1.1 illustrates the concept of multigenic chromosomes; in this instance a 3-genic specimen.

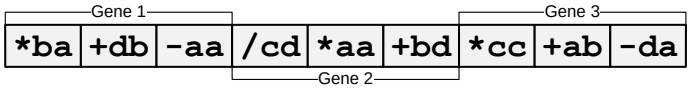

Figure 3.6: Sample multigenic symbolic regression chromosome with 3 genes

3.1.2 NEXT and Expression Trees

At its heart, NEXT is a methodology for evolving solutions with an expression tree-based phenotype. What is unusual about NEXT is the way it deals with the representation of these trees. NEXT chromosomes represent tree structures in a hierarchical-linear form, and the mapping to the tree phenotype is implicit, rather than resulting in the explicit construction of trees.

Diagrams of expression trees appear throughout this work, and indeed it is easy to sketch such a tree upon inspection of a given NEXT chromosome. The process for creating such a tree within the system, however, can be quite computationally costly and was avoided here in favour of a linear method of evaluation described in Section 3.1.3.

The concept of the implicit expression tree (ET) can be better understood through considering an example. Figure 3.7 contains a simple NEXT gene with four cistrons. Each cistron consisting of one function and two terminals, as the maximum arity of the functions used in

mathematical modeling problems is assumed to be two.

In a fashion similar to the genotype-phenotype translation employed in GEP, each function in the gene takes as its children the function(s) to its right until the arity is satisfied or there are no more functions left. The pseudo-ET in Figure 3.8 provides a rough illustration of the hierarchical order of the cistrons in Figure 3.7 as they would be ordered in an expression tree.

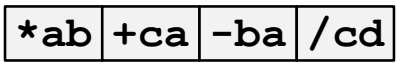

Figure 3.7: A simple NEXT gene

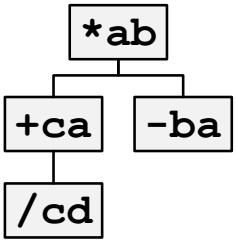

Figure 3.8: Pseudo-ET for 3.7

Figure 3.9 illustrates the gene in Figure 3.7 as it would appear in a more conventional expression tree. During the process of the interpretation of the gene to the phenotype, expressed terminals become the leaf nodes of the tree and those which are not needed (introns) are bypassed.

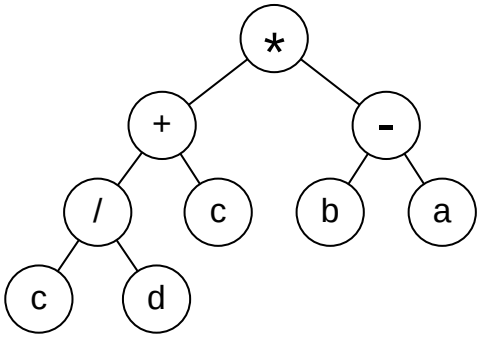

Figure 3.9: Conventional ET for Figure 3.7

3.1.3 Evaluation of NEXT Chromosomes

The structural organization of NEXT is such that the GPM can be accomplished without having to construct an intermediate tree or matrix data structure to assist the evaluation a chromosome. The evaluation of NEXT genes is conducted in two passes. In the first pass, the gene is read

from left to right to determine the assignment of children nodes. During the second pass the gene is read right to left, evaluating each cistron individually.

This linear evaluation is made possible because any given cistron can be evaluated if the values of its children are known. The rightmost cistrons of a given gene contain the terminals of the expression tree, the values of which are already known as the input to the expression. As the evaluation proceeds leftward, the values of child cistrons are computed before their parents. In this fashion, evaluation can be conducted without the need for the expression tree to be constructed and traversed.

The process of evaluation may appear rather complex compared to conventional methods, but is actually quite simple. Figure 3.10 contains a simple NEXT gene for a mathematical expression consisting of four cistrons. For illustrative purposes, the cistrons are numbered `C0` through `C3`.

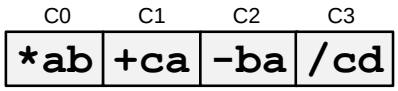

Figure 3.10: Simple four-cistron gene

The first phase of evaluating a NEXT gene is to determine the assignment of the child cistrons. The cistron is read from left to right, and the arity of each function checked. All functions in this example have an arity of two, and thus require two inputs. Cistron `C0` is assigned the values of `C1` and `C2` as its inputs. Proceeding rightward, `C1` requires two inputs but as only `C3` remains unassigned, the first terminal within `C1` will be used as the second input. The remaining cistrons have no available children, and thus as leaf nodes will use their terminals. The assignment of children in this example is shown in Figure 3.11.

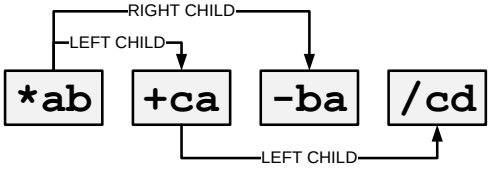

Figure 3.11: Simple four-cistron gene

Figure 3.12 shows the tangible result of this child assignment, in the form of the simple

calculations required for each cistron. We can see that only C2 and C3 can be immediately evaluated without prior calculations being performed.

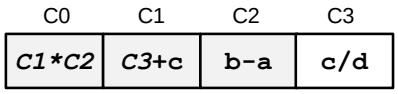

Figure 3.12: Equations for each cistron

With the child cistron assignment complete, the second phase of the evaluation process may begin and the values of individual cistrons calculated. As the structure is such that any cistron can be evaluated knowing the value of its inputs, the evaluation begins with the rightmost cistron and proceeds leftward. For this example, the terminals of **a,b,c,d** are assigned the simple integer values of **1,2,3,4** respectively.

Figure 3.13 illustrates the right-to-left evaluation of this gene in steps, proceeding from **C3** to **C0**. At each step, the cistron being evaluated is shaded in grey, and the evaluation being performed listed at right.

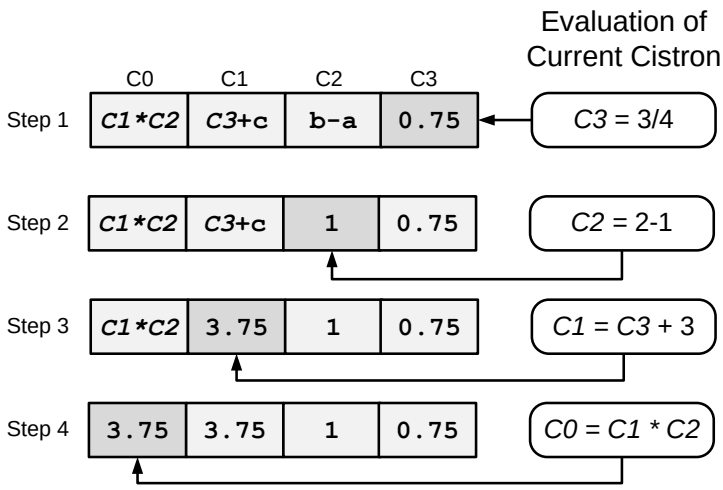

Figure 3.13: Right-to-left evaluation of gene, step by step

We can see that for **C3** and **C2**, evaluations take place using only the terminal inputs. For the remainder of the gene, terminal inputs are combined with the calculated values for the child cistrons. The result of the evaluation is at **C0** following the completion of the evaluation. Were this gene part of a population during a run, that value would be used to compute the fitness measure of the chromosome (individual).

It is notable that a genotype-phenotype mapping has taken place implicitly during this process, and the cistrons have been evaluated in tree form without the computational cost of constructing trees, or evaluating trees recursively as is required in other tree-based methods. This operation is linear based on the two passes across the cistrons of the gene.

3.1.4 Genetic Operators

Genetic operators are the means by which individuals in the population undergo genetic exchange and modification with the aim of obtaining higher fitness. These operations take their inspiration from the biological mechanisms through which living organisms evolve and reproduce. While certainly the precise mechanisms of biological reproduction are not replicated in evolutionary computation systems, the concepts are mimicked within the constraints of a simplified genome.

The operators in NEXT fall under two categories: those which operate within a given entity and those which operate between entities, termed Intra-Entity and Inter-Entity. Operations can also take place at each structural level, ranging in scope from simple point mutations at the chromosome level to gene transposition and insertion. The operations used can vary The structure of NEXT chromosomes allows for a great many genetic operators to be developed, and what is implemented in this system is only a subset of the potential operators which could be developed by applying the same principles to more levels of the genetic hierarchy.

The genetic operators presented here are in order of the level at which they occur.

Point Mutation

Point mutation is the lowest level operator included in NEXT, taking direct inspiration from the biological point mutation mechanism. In the biological mechanism, point mutation occurs when one nucleotide is substituted for another. In NEXT terms, point mutation selects one position within a given cistron and substitutes the codon at that location for another randomly selected codon. The position selected may be a terminal or a function, but the structure of

having one function per cistron is enforced and codons may only be substituted for one of their own type.

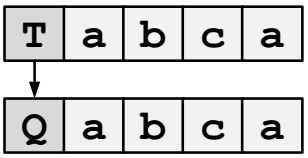

Figure 3.14: Point mutation on a function

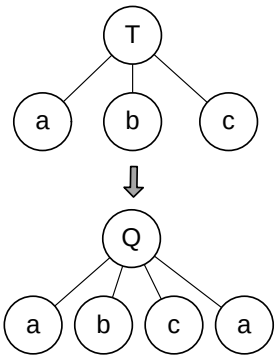

Figure 3.15: Sub-ETs resulting from this mutation

Figure 3.1.4 illustrates a point mutation operation on a function, where the neural function **`T`** is substituted for the function **`Q`** representing a change in arity from 3 to 4 and the addition of one terminal. The results of this point mutation can be seen in Figure 3.1.4.

Point mutations can also occur at positions with terminals, as shown in Figure 3.1.4 where the terminal **`a`** is mutated to **`d`** through the operator. The resulting sub-expression tree is not structurally changed, so the visual representation is not shown. The operation simply results in a different input for the neural network being evolved.

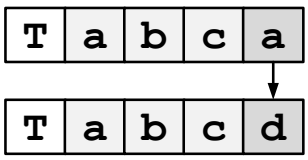

Figure 3.16: Point mutation on a terminal

Cistron Insertion

The Cistron Insertion operator inserts a new randomly generated cistron at a randomly selected portion of the gene, increasing the size of the gene by one cistron. The insertion operator takes its inspiration from the biological mechanism where one or more nucleotides are inserted into the DNA sequence of an organism.

The motivation behind the inclusion of this operator is to allow for concepts of complexification to be realized in NEXT by allowing genes to start relatively small, thus minimizing the search space. By inserting more cistrons as evolution progresses, the search space can grow until a level is reached where the optimal solution resides.

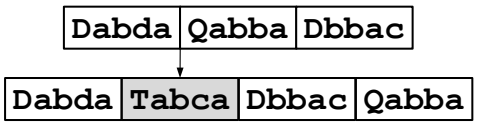

Figure 3.17: Cistron insertion

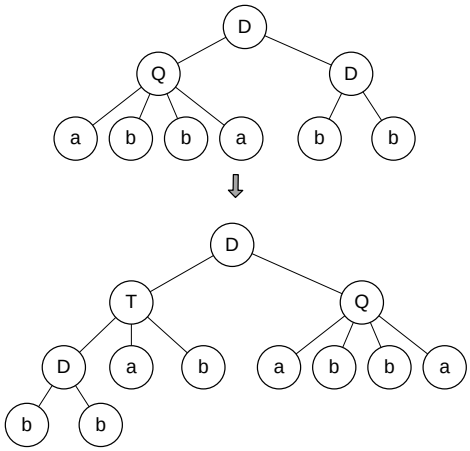

Figure 3.18: Expression trees for Figure 3.17

Figure 3.17 illustrates the insertion of a new cistron **`Tabca`** at index 1 of the gene. The effects of this insertion are visualized in expression tree 3.18 where the additional sub-expression tree can be seen. Due to the order in which NEXT genes are evaluated and trees constructed, the sub-ET representing **`Dbbac`** becomes a child of the new sub-ET and increases the height of the tree.

Noteworthy in this operation is that the previous substructures are retained in whole, such as the sub-ETs for **`Dbbac`** and **`Qabba`**, allowing that genetic information to remain. This is important because a given substructure may be highly valuable to the solution, and a method of expanding the search space without destroying already viable structures is essential to the ability of the system to produce good solutions.

Cistron Transposition

The cistron transposition operator selects two cistrons from random positions within the same gene and exchanges them. The motivation behind such an operator is to introduce a means for structural variation to take place at the function level while retaining the relationships between

a function and its terminals.

This operator is especially useful in symbolic regression problems where a cistron transposition represents a change to the function due to order of operations.

Figure 3.19 contains an illustration of cistron transposition acting on the genotype, with the two cistrons changing positions within the gene. The effect of this transposition on the phenotype can be seen in Figure 3.20.

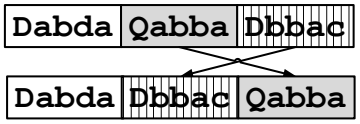

Figure 3.19: Cistron transposition

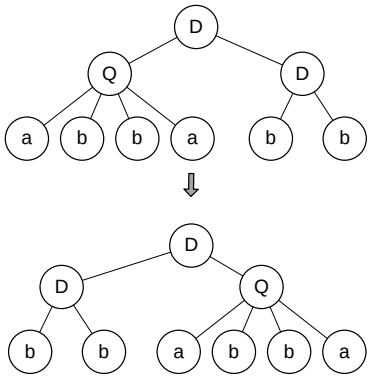

Figure 3.20: Expression trees for Figure 3.19

Cistron Transfer

The purpose of the cistron transfer operation is to shift genetic material between chromosomes and enforce diversity in the size of individuals. The operator randomly selects a cistron from within a given chromosome, and inserts it at a random location in another chromosome. The two chromosomes in question are also selected randomly.

Figure 3.1.4 illustrates the cistron transfer operation as it would affect the genotype. The cistron **Qabba** is removed from the second position of Gene 1 and inserted at the third position of Gene 2.

Figure 3.21: Cistron transfer

Figure 3.1.4 shows the phenotypic effect of this transfer. Prior substructures such as **Dbb** in

Gene 1, and **Ua**, **Dbd** in Gene 2. The subtree **Qabba** also remains intact and is simply inserted into Gene 2, transferring its functionality between genes.

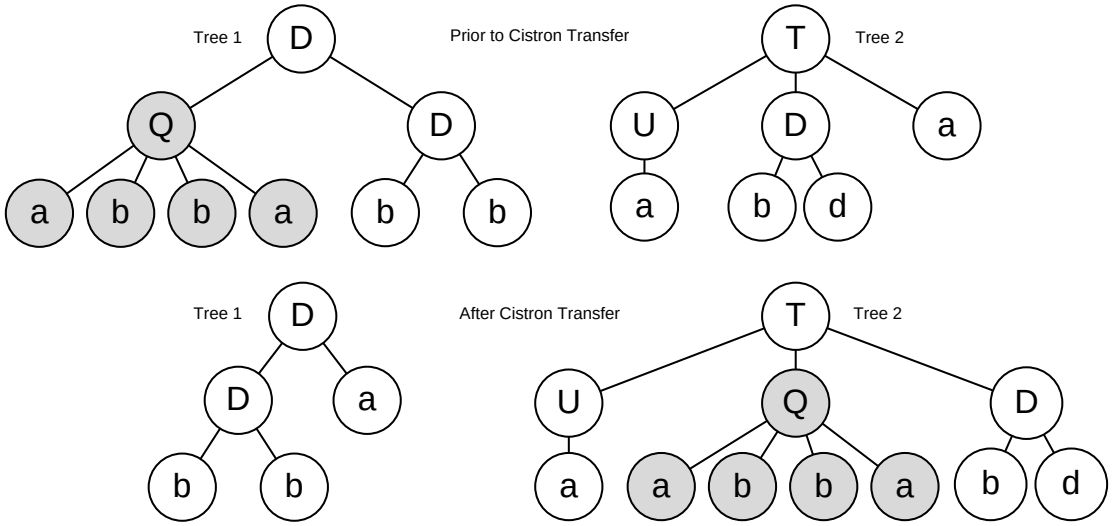

Figure 3.22: Expression trees for Figure 3.1.4

Recombination

Recombination (crossover) has been included in many evolutionary computation systems, Genetic Programming and GEP included. In GEP, recombination takes place by lining up genes of identical size and swapping a given position. As NEXT contains individuals of disparate sizes, exchanging the same position is not possible.

The recombination operator in NEXT works by way of swapping a randomly selected cistron in one randomly selected gene for a randomly selected cistron in another gene. This is somewhat similar to the way crossover operates in GP, but does not swap entire subtrees, rather one node and any terminals it may have as children.

Figure 3.1.4 depicts two genes with one cistron in each randomly selected for recombination. Note that the selected cistrons are not at the same position within their respective genes.

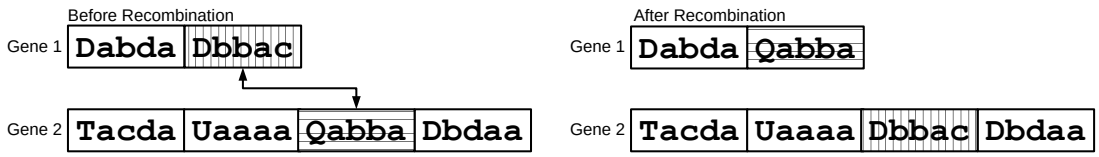

Figure 3.23: Cistron recombination

Figure 3.1.4 illustrates the phenotypic result of this exchange. As both selected cistrons contained leaf node terminals, the subtrees were simply swapped between trees with no other changes taking place.

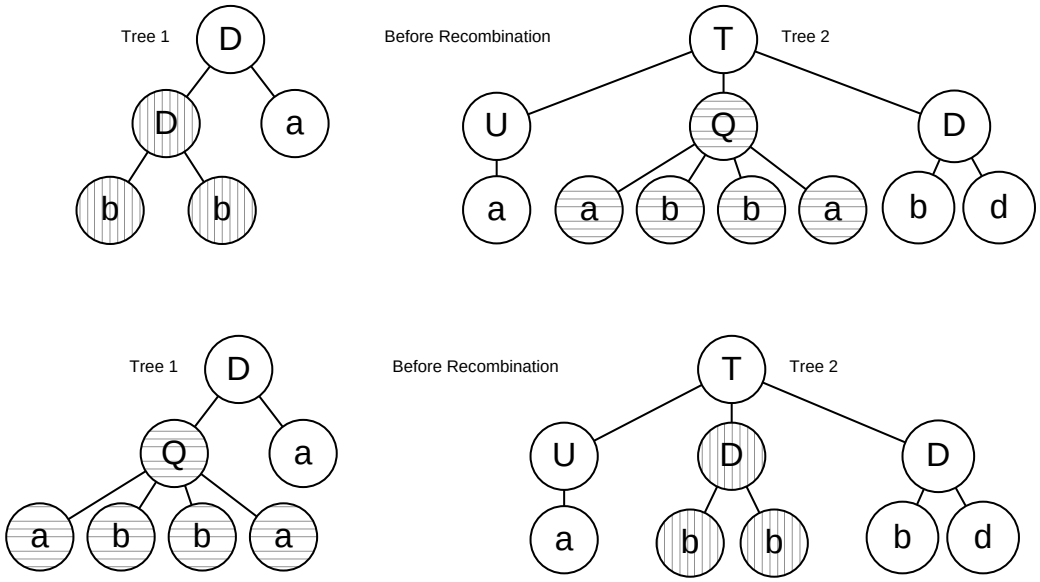

Figure 3.24: Expression trees for Figure 3.1.4

3.1.5 Neuroevolution Operators

Neuroevolution problems require additional operators to maintain the weights and thresholds used in the network. The three types of operators used in NEXT for this purpose are:

Weight Mutation

The weight mutation operator randomly selects one weight within a given cistron and replaces its value with a new randomly enerated floating point number.

Weight Transposition

Weight transposition randomly selects two weights within a given cistron and transposes their positions, providing for a change in the weighting of inputs.

Threshold Mutation

Threshold mutation is the primary mechanism by which thresholds are modified in NEXT neuroevolution cistrons. When a given cistron is selected for threshold mutation to take place, the threshold is replaced with a new randomly selected floating point value.

Threshold Transposition

Threshold transposition is applied less frequently than threshold mutation due to its slightly more destructive nature. In this operation, two cistrons within a given chromosome are randomly selected and their thresholds transposed.

3.1.6 Creation of the Initial Population

As the optimal chromosome size is not known *a priori*, the initial population must contain solutions of varying sizes to allow the system to work towards optimal dimensions. The method of population creation includes a mechanism for seeding the population with genes of random sizes between one cistron and a specified maximum. This method results in an initial generation of individuals of varied size, and provides greater genetic diversity during the initial runs.

3.1.7 Selection Method

The selection method employed by NEXT is a common roulette-wheel [13] method with proportional selection. Each individual in the population is assigned a portion of the selection space proportional to its contribution to fitness. During the selection phase of the evolutionary algorithm, the mechanism selects individuals according to the specified population size. This method theoretically rewards high fitness through a greater proportion of the space on the roulette wheel being allocated to individuals of higher fitness.

Elitism is also utilized wherein the most fit individual of a given generation is automatically copied to the new population outside of the selection mechanism.

3.1.8 The Evolutionary Algorithm of NEXT

The methods of evaluation and genetic modification discussed in this chapter are combined as a process to evolve NEXT chromosomes towards a viable solution. Figure 3.25 illustrates the evolutionary algorithm by which these operations take place.

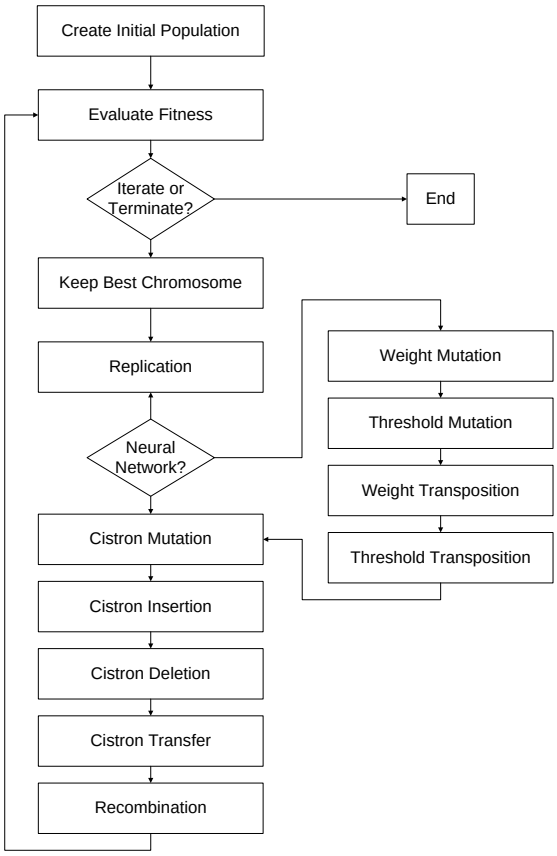

Figure 3.25: NEXT evolutionary algorithm

A run with the NEXT system begins with an initial random population is created by the method described in Section 3.1.6. The fitness of each individual within the population is then evaluated according to the fitness function in use for the problem in question.

At the completion of fitness evaluation the decision is made to either complete the run, having found an ideal solution or continue. If continuing, the system begins its process of creating a new population from the existing one through genetic operators. The best solution from the previous generation is saved as is, and replication selects the constituents of the new population according to their fitness.

If the problem at hand is one of neuroevolution, the operators for modifying the weights and

thresholds are applied in order. Following this, the operations of mutation, insertion, transfer, and recombination are applied before the process iterates again.

3.2 Experimental Design

This section outlines the experimental design employed in the testing and validation of NEXT as an EC system. Included are descriptions of the problems used, the conditions under which the experiments were conducted, and the aims of said experiments.

The problems used in demonstrating NEXT were selected to highlight the capabilities of NEXT for both mathematical modeling and neuroevolution. Intended to benchmark NEXT against GEP, many of the problems selected are those which Ferreira has used to demonstrate GEP [7] and GEP-NN [8]. Included are several demonstrations of symbolic regression problems of increasing complexity, as well as methods of logic synthesis and pairwise classification using evolved neural networks.

The problems are intended to illustrate several aspects of the performance of NEXT. It will be shown that NEXT can evolve high quality solutions for problems in the above domains. Of greater importance however, is the demonstration of consistency of results. Given that all EC systems are essentially stochastic processes producing large populations of random individuals, it is possible for a single good solution to be developed in one run by nothing more than fluke.

The tests run here should demonstrate consistency over many runs of several factors: high fitness, high average fitness of the population, consistency in the size of the chromosome (ie: bloat does not take place), and a relatively consistent number of generations to evolve a solution for a given problem.

3.2.1 Runs, Sets of Runs, & Validation

In the terminology of NEXT, a *run* is one completion of the evolutionary process, terminating either with the desired fitness or after a fixed number of generations. A *set of runs* is quite simply a collection of these runs completed in order to assure validation of results.

All experiments conducted were completed under 10-fold validation, with ten sets of ten runs being completed, and averages taken to assure consistent results. In the case of the classification problems, the data set was divided into 10 ”folds”, and for each run in a given set, was rotated with a different set used as the validation set for each run.

3.2.2 Symbolic Regression

Symbolic regression, as the term is used in the EC space, is the automatic finding of a mathematical function from a set of data points. The experiments contained herein use as input a series of data points from the equation given, and over a number of iterations the system attempts to reconstruct the original equation.

The NEXT system parameters used in the following four experiments are summarized in Table 3.1.

Table 3.1: Symbolic regression experiment parameters

Parameter	Exp. 1	Exp. 2	Exp. 3	Exp. 4
Number of Generations	100	100	100	2000
Number of Runs	10	10	10	10
Population Size	100	100	100	100
Initial Maximum Gene Size	1	1	1	1
Point Mutation Rate	0.05	0.05	0.05	0.05
Cistron Insertion Rate	0.05	0.05	0.05	0.05
Cistron Transfer Rate	0.1	0.1	0.1	0.1
Cistron Mutation Rate	0.1	0.1	0.1	0.1
Recombination Rate	0.1	0.1	0.1	0.1

Experiment 1: Simple Symbolic Regression for Validation

The first problem to be demonstrated is the simple exponential function shown in Equation 3.1.

$$f(x) = x^5 \tag{3.1}$$

This problem was selected for the purpose of demonstrating that NEXT is capable of evolving a gene of optimal size for a simple mathematical expression without any extrane-

ous cistrons, that is to say cistrons which do not contribute to the fitness and serve only to expand the size of the solution. The input values were integers on the range [1,10].

Experiment 2: Multi-variable Function

Equation 3.2 is a simple function of two variables intended to demonstrate the abilities of NEXT with regards to symbolic regression with multiple variables and integer constants. As NEXT does not have facilities for more complex evolution of constants, this experiment will include a terminal representing the integer `1` to facilitate the creation of the constants required and may also rely on arithmetic means to compose constants.

$$y = a^2 + b^2 + 2a + 2b \tag{3.2}$$

The values used in this problem are listed in Table 3.2.

a	**b**	**y**
1	0.1	3.21
2	0.2	8.44
3	0.3	15.69
4	0.4	24.96
5	0.5	36.25
6	0.6	49.56
7	0.7	64.89
8	0.8	82.24
9	0.9	101.61
10	1	123

Table 3.2: Experiment 2 data

Experiment 3: Fourth Order Polynomial

This problem was used by Ferreira to demonstrate the symbolic regression capabilities of GEP. [7] It is included here to highlight the abilities of NEXT to evolve higher order polynomials. The equation governing this experiment is stated in 3.3.

$$f(x) = x^4 + x^3 + x^2 + x \tag{3.3}$$

The data set used in this trial is the same as was used in the GEP paper, and is listed in Table 3.3.

x	**f(x)**
2.81	952.425
6	1554
7.043	2866.55
8	4680
10	11110
11.38	18386
12	22620
14	41370
15	54240
20	168420

Table 3.3: Experiment 3 data

Experiment 4: Sequence Induction

This problem was also used by Ferreira in demonstrating GEP. [7] and is included here to demonstrate the capabilities of NEXT with regards to integer constants. As in Experiment 3, constants are produced through the use of the integer **1** as a terminal and addition thereof.

The sequence to be induced is goverened by Equation 3.4 where n is the order in the sequence. The input data used in this experiment is listed in Table 3.4.

$$N = 5a_n^4 + 4a_n^3 + 3a_n^2 + 2a_n + 1 \tag{3.4}$$

Table 3.4: Experiment 4 data

a	**N**
1	15
2	129
3	547
4	1593
5	3711
6	7465
7	13539
8	22737
9	35983
10	54321

3.2.3 Neuroevolution for Logic Synthesis

NEXT is capable of evolving neural networks for a variety of problem types. Logic synthesis was selected as a classical example of the type of problem used in demonstrating neuroevolution systems.

Experiment 5: XOR

This problem was used by Ferreira [8] to demonstrate the abilities of GEP-NN to evolve neural networks for logic synthesis. The system is given the truth table for XOR, and expected to evolve a network representing the function.

The input consists of the truth table for XOR as listed in Table 3.5 and the system parameter set listed in Table 3.6

Table 3.5: Truth table for XOR

a	b	x
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.6: Experiment 5 parameters

Parameter	Value
Number of Generations	100
Number of Runs	10
Population Size	100
Initial Maximum Gene Size	1
Point Mutation Rate	0.05
Cistron Insertion Rate	0.05 Cistron Transfer Rate
0.1	
Cistron Mutation Rate	0.05
One-Point Recombination Rate	0.05
Threshold Mutation Rate	0.05
Weight Mutation Rate	0.18

3.2.4 Neuroevolution for Pairwise Classification

Experience with pairwise classification in GEP-NN was the primary impetus for the development of NEXT, and this experiment should demonstrate gains over GEP-NN based on the improvements made with this variety of neuroevolution in mind.

This set of experiments is designed to demonstrate the neuroevolution capabilities of NEXT for use in developing neural classifiers. The experiments employ a scheme for pairwise classification based upon a method developed in previous work by the author [16] for the development of pairwise neural classifiers [20] with GEP-NN.

The method employs pairwise decomposition as a means to reduce classification problems to a series of binary classifiers. A majority vote combination rule is used to determine the winning class. A multigenic implementation is used, where the number of genes in each chromosome (n) is determined by Equation 3.5. [21]

$$n = \frac{k(k-1)}{2} \tag{3.5}$$

Each chromosome contains n = k(k-1)/2 genes, where k is the number of classes in the problem. Each gene represents one component of the classifier, simply deciding between one of two classes.

Input data is partitioned into subsets for each gene so that each sub-classifier is only pre-

sented with data for two classes. For validation purposes, k-fold cross validation is used, with 10 folds. In each run the k^{th} data subset is used solely as a validation set.

Both experiments used the same system parameters, listed in Table 3.7.

Table 3.7: Experiment 6 & 7 parameters

Parameter	Value
Number of Generations	2000
Number of Runs	10
Population Size	100
Initial Maximum Gene Size	1
Point Mutation Rate	0.05
Cistron Insertion Rate	0.05 Cistron Transfer Rate
0.1	
Cistron Mutation Rate	0.05
One-Point Recombination Rate	0.05
Threshold Mutation Rate	0.05
Weight Mutation Rate	0.18

Two data sets were employed in the classification experiments, the Fisher Iris [10] data set, and the Wine [6] data set. The data sets used were obtained from the UCI Machine Learning Repository [1].

Experiment 6: Fisher Iris Data Set

The Fisher Iris data set [10] consists of 150 records of three classes, with the records representing an even 50 classes each. The data set has four attributes, described in Table 3.8 along with the NEXT terminal used to represent each attribute.

Table 3.8: Attributes of the Fisher Iris Data Set

Attribute	Terminal
Sepal Length in cm	a
Sepal Width in cm	b
Petal Length in cm	c
Petal Width in cm	d

Experiment 7: Wine Data Set

The Wine data set is intended to distinguish between varietals of wine based upon chemical analysis results of three distinct types of wine. The data set contains thirteen attributes, as listed in Table 3.9.

Table 3.9: Attributes of the Wine Data Set

Attribute	Terminal
Alcohol	a
Malic acid	b
Ash	c
Alcalinity of ash	d
Magnesium	e
Total phenols	f
Flavanoids	g
Nonflavanoid phenols	h
Proanthocyanins	i
Color intensity	j
Hue	k
OD280/OD315 of diluted wines	l
Proline	m

Chapter 4

Results and Discussions

This chapter presents the results obtained during trials of the NEXT implementation with a selection of problems. Presented here are the results of problems of symbolic regression to demonstrate the mathematical modeling capabilities of NEXT, as well as problems of logic synthesis and classification to demonstrate the ability of NEXT to induce neural networks.

The results presented herein are discussed in terms of not only the fitness of the solutions obtained, but in light of several other factors essential to the demonstration of such a system. The presentation of results to each experiment will include discussion of the error distribution, chromosome size distribution, and repeatability of results over the course of many runs.

The notation *Run x-y* is used in this chapter to reference a specific run of the system during trials, *ie:* Run 0-4 refers to Set 0, Run 4.

4.1 Symbolic Regression Results

	Best Fitness	Average Fitness	Average Size	Standard Deviation (Size)
Experiment 1	100%	97.8%	4.31	0.774
Experiment 2	100%	99.92%	10.20	3.316
Experiment 3	100%	92.18%	5.46	1.28
Experiment 4	100%	81.18%	17.54	4.529

Table 4.1: Summary of symbolic regression results

4.1.1 Experiment 1: Simple Symbolic Regression

This symbolic regression experiment was devised to illustrate the capabilities of NEXT to produce individuals of ideal size. A simple function of $f(x) = x^5$ was selected for this purpose as

it involved only one variable, and was of minimal complexity.

Ten sets of ten runs were conducted, with an overall average fitness of 97.8%. Three runs were unsuccessful, but the remaining 97 runs ably found an appropriate compact solution.

Figure 4.1 depicts the fitness values obtained during a sample run, in this instance run 0-4. Shown on this chart are the best fitness of the run to the current point, the best fitness of the given generation, and the average fitness from each generation. The best fitness of the run and best fitness of the generation are largely coincident as each generation typically produces a new best fitness.

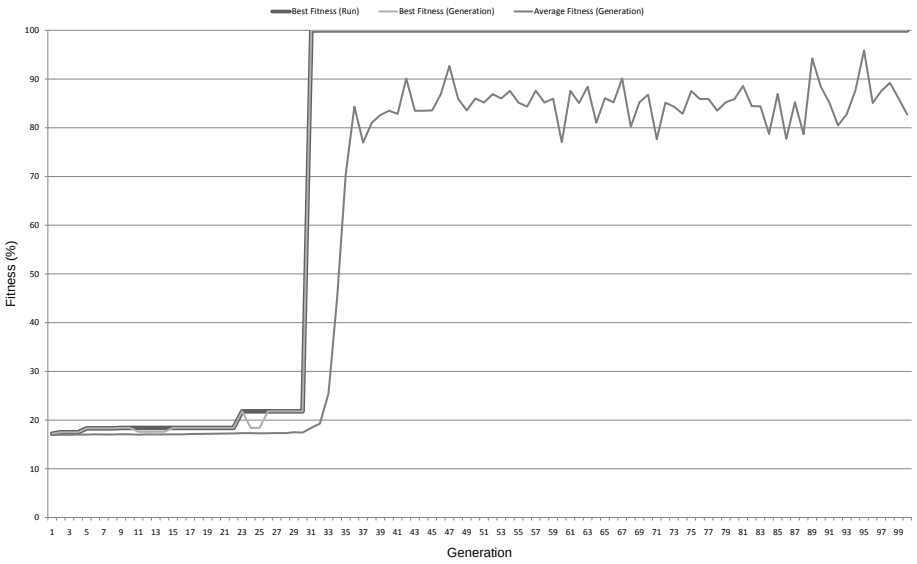

Figure 4.1: Experiment 1: Fitness values - run 0-4

The sudden jump in fitness encountered at generation 31 is typical of the results obtained throughout all runs, and is a result of the growth of the genes reaching a point at which a fit solution can be represented. This can be seen through considering Figures 4.2 and 4.3 side by side, where the gain in chromosome size is coincident with the sudden jump in fitness.

The compact representation shown in Figure 4.4 is representative of the solution found by the majority of runs. The translation of this individual to a mathematical expression can be found in Equation 4.1. It can be seen that this is a compact representation of the solution without extraneous nodes, thus perfectly satisfying the requirements.

$$f(x) = ((a * a) * a) * (a * a) \tag{4.1}$$

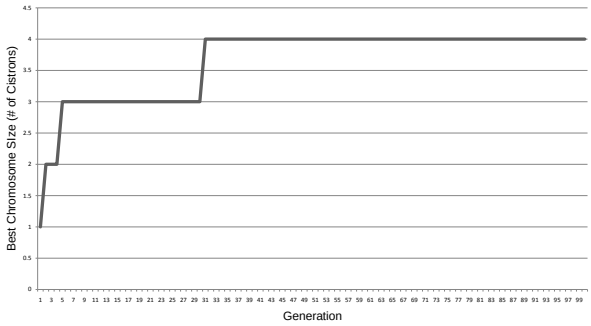

Figure 4.2: Experiment 1: Chromosome sizes by generation - run 0-4

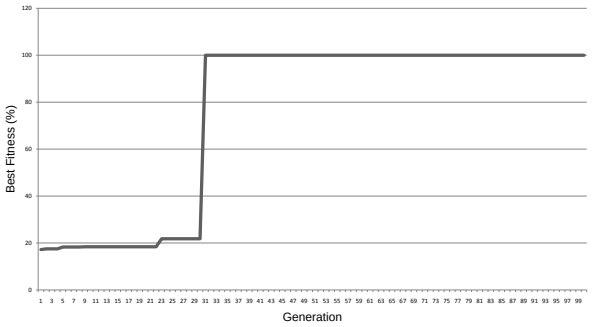

Figure 4.3: Experiment 1: Best fitness by generation - run 0-4

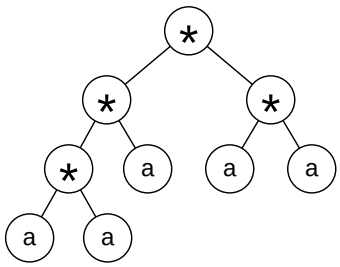

Figure 4.4: Experiment 1: Representative solution

Figure 4.5 shows the best fitness achieved during each run of the 100 runs conducted. Note that these runs were conducted as a series of ten sets of ten runs, but are displayed here as a set of 100 runs for visualization purposes. As previously mentioned, perfect solutions were found for this simple problem in 97 of 100 runs. The remaining runs produced solutions similar to 4.6, where for reasons of insufficient evolutionary change, the population became saturated with individuals without the appropriate functions.

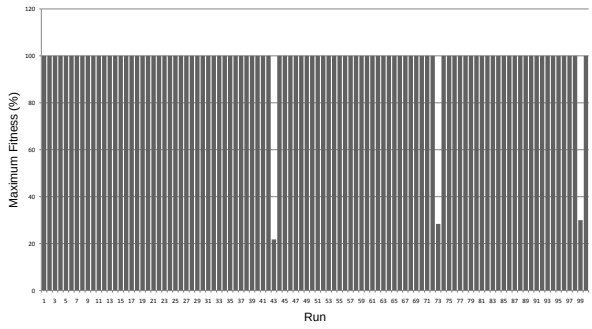

Figure 4.5: Experiment 1: Maximum fitness by run

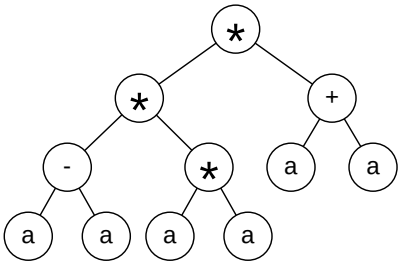

Figure 4.6: Experiment 1: Example of a poor solution

The chromosome sizes encountered during the solution of this problem remained consistent, with an average best chromosome size of 4.31 cistrons and a standard deviation of 0.774

cistrons. Figure 4.7 depicts the size of the best solution by run. It can be seen that the majority of runs experienced chromosome sizes in the vicinity of four cistrons in length. There are outliers present in the range of five to eight cistrons per chromosome, but it is not believed that this pattern indicates a tendency of the system towards bloat.

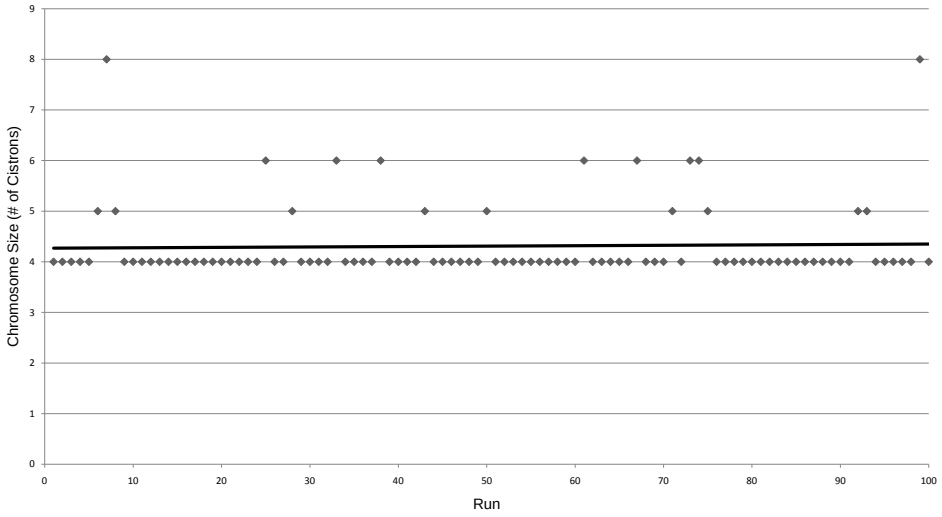

Figure 4.7: Experiment 1: Size of best solution by run

4.1.2 Experiment 2: Multi-variable Function

This experiment was intended to demonstrate the capabilities of NEXT in evolving symbolic regression solutions with multiple variables. Tests were successful, with an average fitness of 99.92% over 100 runs.

Figure 4.8 illustrates the fitness values obtained over the course of a typical run, in this instance Set 1, Run 1. The maximum fitness achieved during each generation and the maximum during the run were once again largely overlapping. During the latter half of the run there were instances where the best fitness of the generation was down a few percent, most likely due to mutations of the best chromosomes. The average fitness of the population during the run followed the progress of the best solution, indicating that the population as a whole was evolving towards workable solutions.

Figure 4.9 shows a representative solution found, in this instance the best solution of run 0-5. This solution contains no extraneous cistrons, and represents a perfectly parsimonious

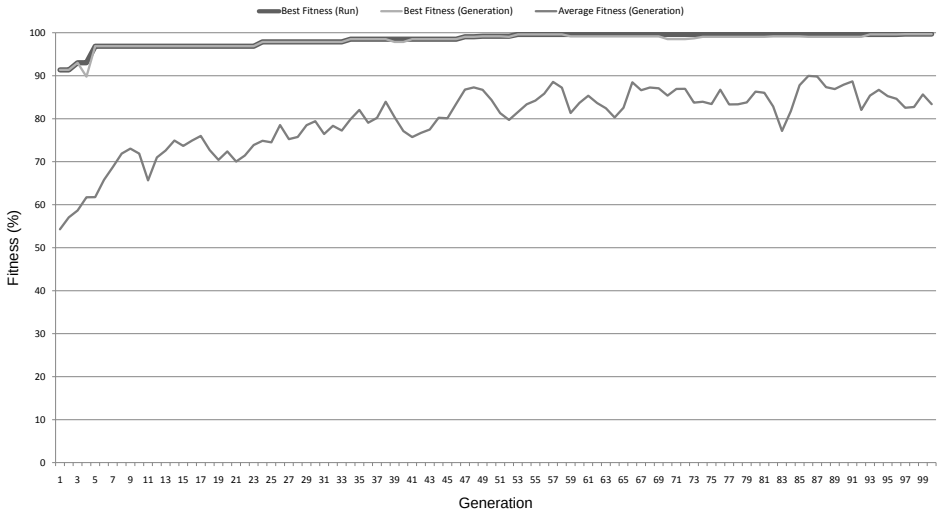

Figure 4.8: Experiment 2: Fitness values - run 1-1

solution to the regression problem at hand.

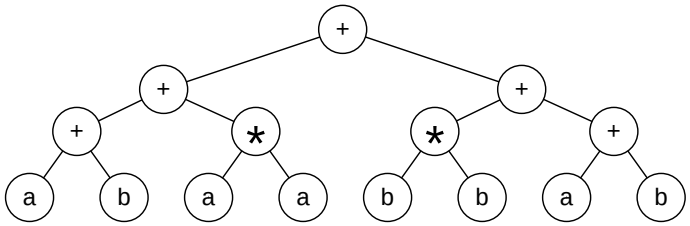

Figure 4.9: Experiment 2: Representative solution

Equation 4.2 contains the equation this phenotype represents, and it can be seen that all four terms of the equation are represented. The *2a* and 2b terms are represented in this equation by the two *(a+b)* terms, as no integer constants were used in this experiment.

$$f(x) = [(a+b)*(a*a)] + [(b*b)+(a+b)] \tag{4.2}$$

The average size of chromosomes during the run was not ideal in the sense of producing entirely parsimonous solutions. The ideal solution presented above contained seven cistrons, while the average of the population was 10.29. This is likely due to the absence of integer constants, which required the composition of such constants through mathematical expressions, increasing the size of the chromosome.

Figure 4.10 illustrates the chromosome size of the best solution by run.

The fitness values obtained in each of the 100 runs conducted are illustrated in Figure 4.11.

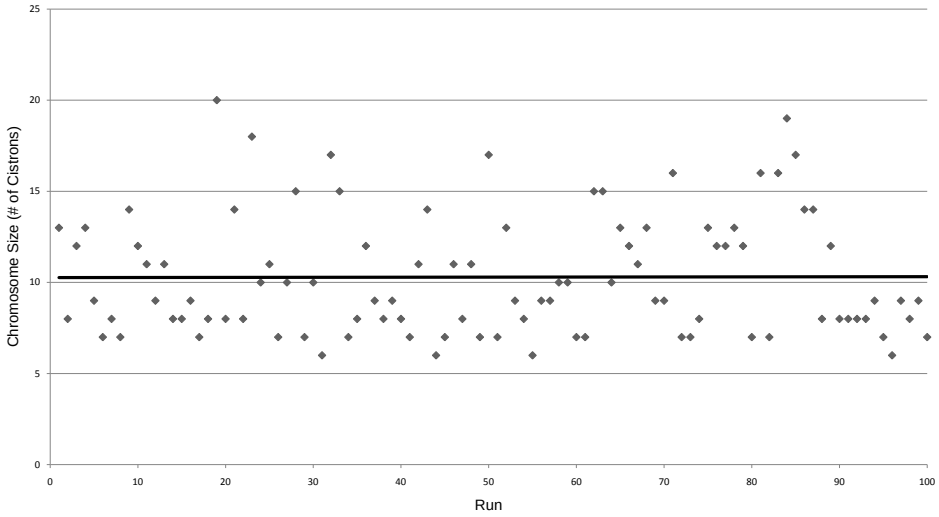

Figure 4.10: Experiment 2: Size of best solution by run

Note that the scale on this chart is from 98% to 100% fitness, so the overall fitness results were very high. The error illustrated of less than one percent took place in instances where more complicated individuals evolved, so minute errors were possible due to the additional terms and small values for **b**.

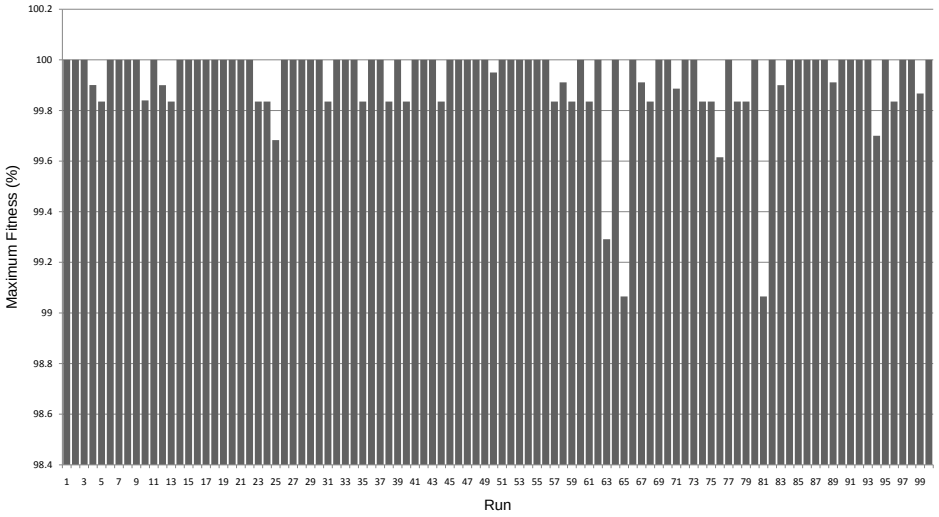

Figure 4.11: Experiment 2: Maximum fitness by run

4.1.3 Experiment 3: Higher Order Polynomial

This experiment required the evolution of a fourth-order polynomial from the data set provided in the demonstration of GEP [7]. The maximum fitness obtained during this experiment was

100%, with an average fitness over 100 runs of 92.18%. GEP was slightly more successful at this problem with an average fitness of 100% over 100 runs.

Figure 4.12 illustrates the fitness values obtained during a sample run which achieved 100% fitness. Shown are the maximum fitness of the run, the best fitness of the given generation, and the average fitness of the generation. The average fitness values of the population once again followed the upward trends of the best fitness, indicating dispersal of effective genetic material throughout the population.

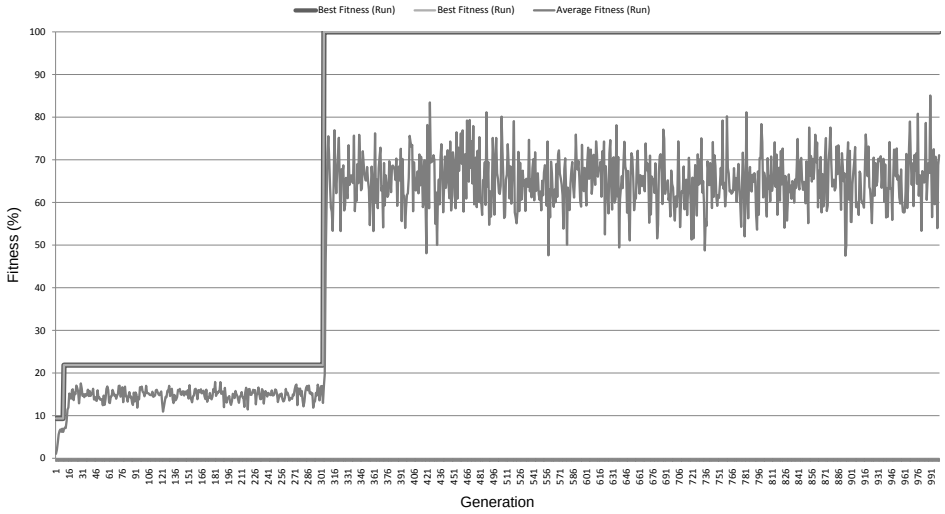

Figure 4.12: Experiment 3: Fitness values - run 1-3

The large jump in fitness seen at generation 304 is once again attributable to a variation in length of the best chromosome, adjusting to suit the requirements of the problem. Figures 4.13 and 4.14 illustrate this correlation. The size of the best chromosome increased to 6, and a perfectly parsiminous solution was found.

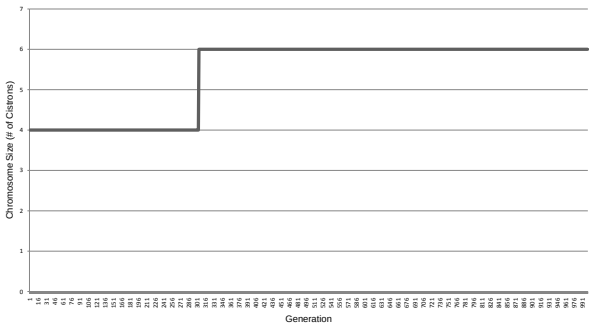

Figure 4.13: Experiment 3: Chromosome sizes by generation - run 1-3

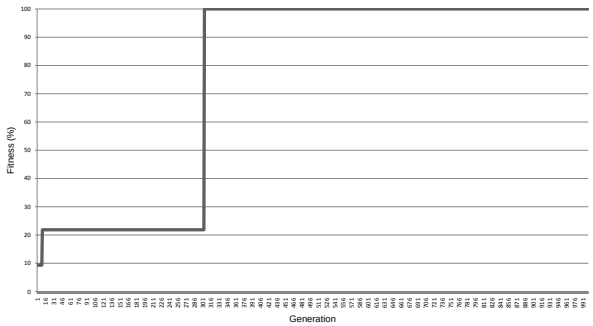

Figure 4.14: Experiment 3: Best Fitness by generation - run 1-3

Figure 4.15 illustrates the solution found at generation 304 of run 1-3, and from the mathematical translation of this phenotype seen in Equation 4.3, it was ideally composed. A simplification of the NEXT-evolved function through expansion reveals that it is indeed the target function.

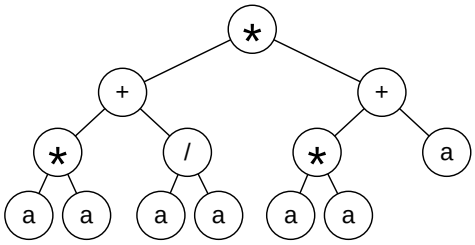

Figure 4.15: Experiment 3: Representative solution

$$f(x) = [(a*a)+(a/a)]+[(a*a)+a] = a^4+a^3+a^2+a \tag{4.3}$$

The chromosome size of the best solution was heavily distributed around the ideal size of 6, with some outliers in runs which obtained poor fitness. This distribution is shown in Figure 4.16.

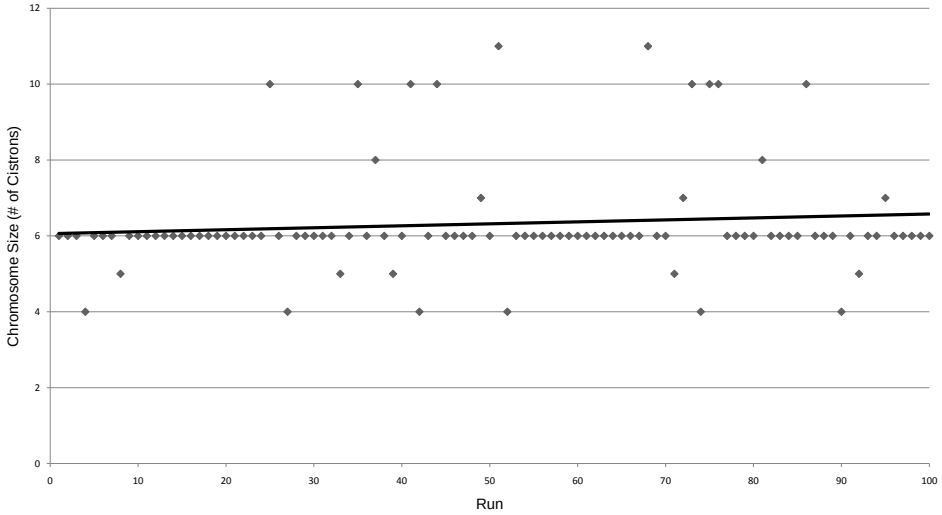

Figure 4.16: Experiment 3: Size of best solution by run

Figure 4.17 depicts the maximum fitness achieved over each of the 100 runs conducted. In seven of the 100 runs, extremely poor final fitness was noticed as was the case in run 0-3 which achieved a maximum fitness of 21.86%. Examination of the best result of this run reveals that

the best individual simply did not contain enough cistrons to produce an effective solution to the problem. Figure 4.18 illustrates the best solution found. Prior best individuals were larger, but with the incorrect functions, and a local minima appears to have been reached by the system and this on other runs with poor fitness.

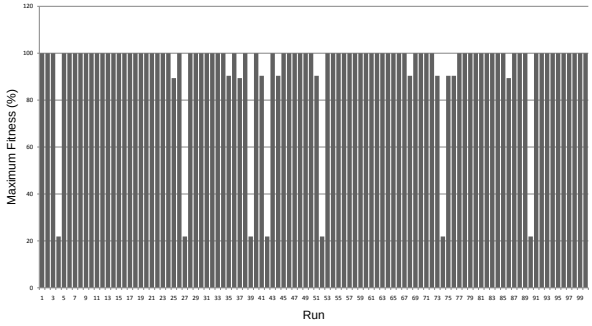

Figure 4.17: Experiment 3: Maximum fitness by run

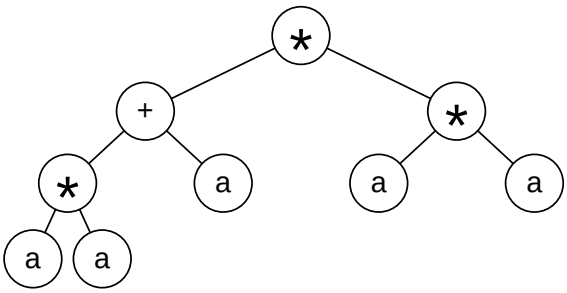

Figure 4.18: Experiment 3: Example of a poor solution - run 0-3

4.1.4 Experiment 4: Sequence Induction

This experiment was not as successful as others performed in terms of achieving consistently high fitness. The average fitness of 100 runs conducted was 81.18%, with a maximum fitness of 100% occuring in 6 runs. GEP was again more successful at this problem than our implementation of NEXT was, achieving an average fitness of 99.9% over 100 runs.

Figure 4.19 depicts the fitness values obtained during a run which was successful. The spike in both the average and maximum fitness ocurring at generation 806 cannot be explained in terms of chromosome size as has been the case with other experiments, as the best individuals of generations 805 and 806 both contained 31 cistrons.

Figure 4.20 shows the chromosome size of the best individuals of each of the 100 runs conducted. It is interesting to note that the larger chromosomes were actually the more fit individuals in this problem. The relation between larger chromosome sizes and higher fitness, despite the fact that these larger chromosomes are statistical outliers accounts for the low average fitness of runs during this experiment. This type of result is not entirely unexpected for this problem, however, as the required composition of several constants for the function requires

much larger chromosomes.

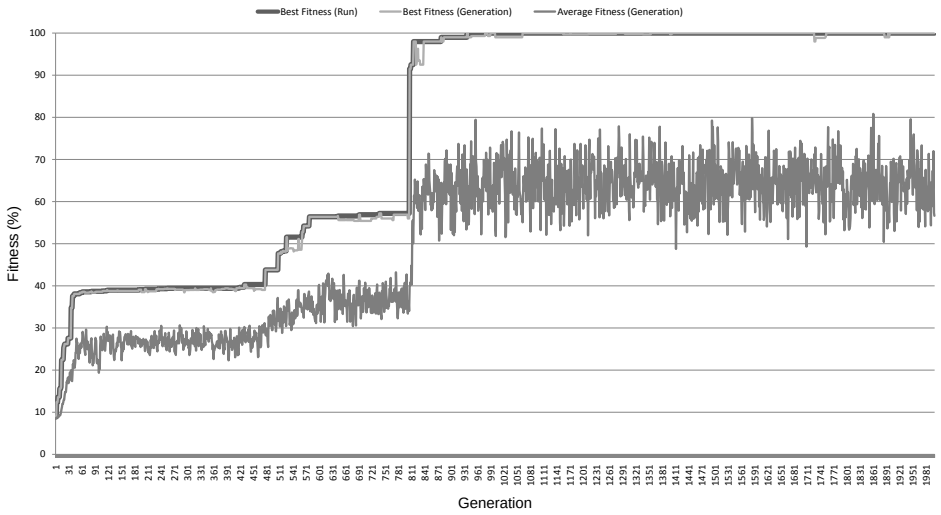

Figure 4.19: Experiment 4: Fitness values - run 0-7

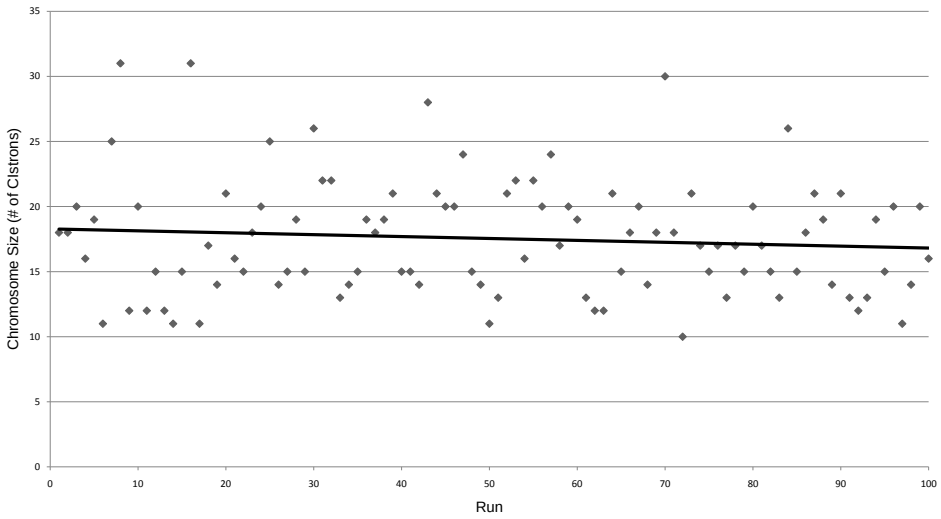

Figure 4.20: Experiment 4: Size of best solution by run

The maximum fitness achieved in each run conducted can be seen in Figure 4.21. The number of runs with low fitness can be attributed to the inability of the system to evolve integer constants for this particular problem.

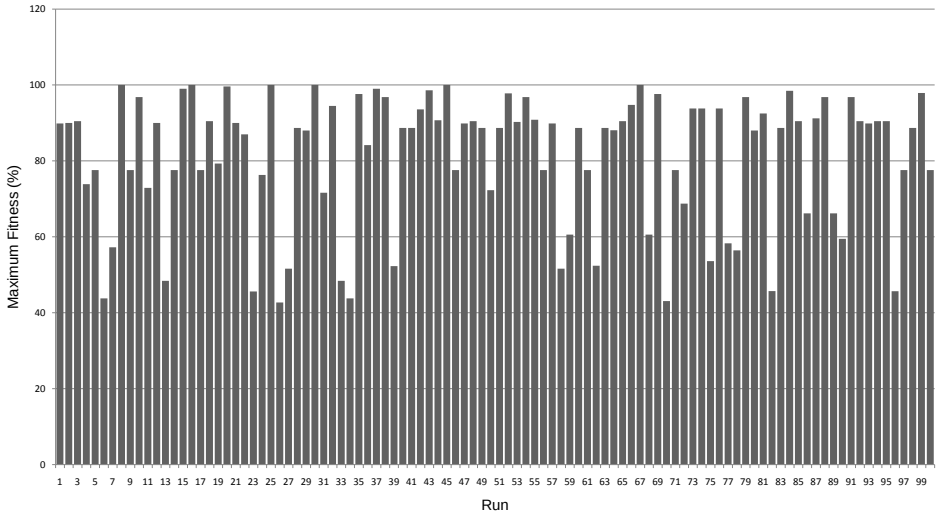

Figure 4.21: Experiment 4: Best fitness by run

4.2 Neuroevolution for Logic Synthesis

4.2.1 Experiment 5: XOR

The results for the induction of a neural network representing XOR were very encouraging. Each run in the set of 100 achieved 100% fitness. In this instance, GEP was not as successful as NEXT, achieving 100% fitness with only 79% probability, compared to the 100% success rate achieved by NEXT.

The fitness values obtained during run 1-1 are shown in Figure 4.22 as representative of a typical run.

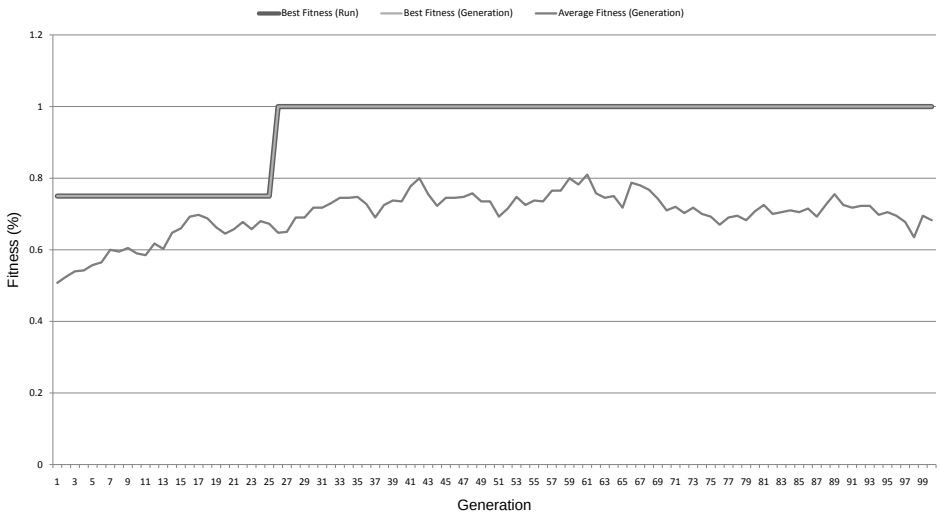

Figure 4.22: Experiment 5: Fitness values - run 1-1

Figure 4.23 shows a sample solution obtained during run 1-1 of this experiment.

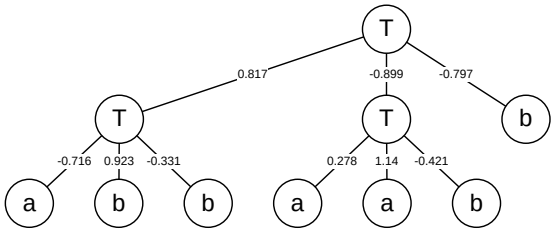

Figure 4.23: Experiment 5: Representative solution

The chromosome size of the best individuals in the population during these runs are shown in Figure 4.24. The chromosome sizes found in the population are biased towards the three cistrons found in the ideal individual shown above. The high level of fitness found in the results of this experiment may be attributable to evolution to a consistent and adequate chromosome size.

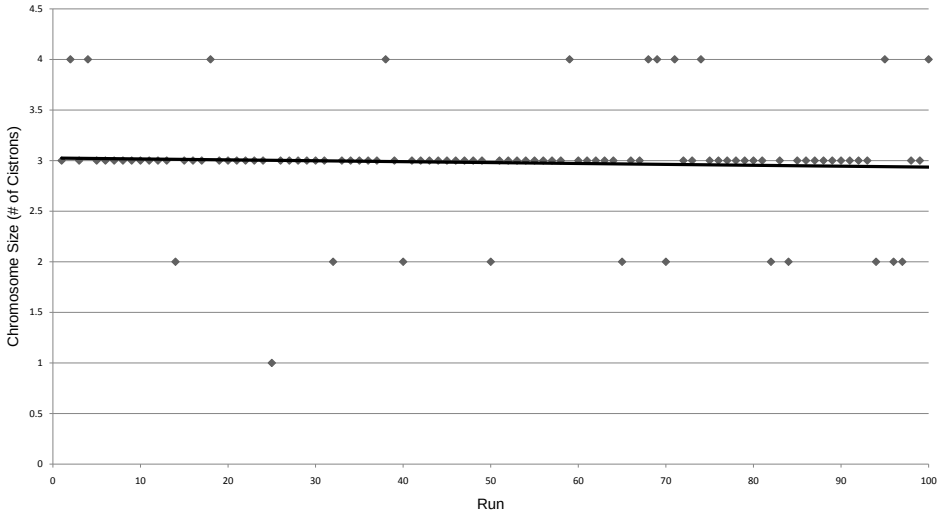

Figure 4.24: Experiment 5: Size of best solution by run

4.3 Neuroevolution for Pairwise Classification

4.3.1 Experiment 6: Fisher Iris Data Set

This experiment proposed to classify the well known Fisher Iris data set using the pairwise classification method previously employed by the author [16]. The results presented here were

obtained over ten sets of runs, in a 10-fold cross validation fashion with one subset of the data being used as the validation set only for each run.

Table 4.2 presents the results obtained from these trials in comparison to those from the author's earlier work with GEP-NN using the same pairwise method. Both values represent the average classification rate of ten sets of ten runs with a rotating validation set, as they were both performed using the same classification methodology. Note that the results obtained by NEXT represent a 4.58% improvement in classification performance over those obtained by GEP-NN.

Table 4.2: Experiment 6: Classification results

Method	Classification Rate	Number Classified
NEXT	98.8%	(14.82/15)
GEP-NN [16]	94.47%	(14.17/15)

Figure 4.25 depicts the fitness values encountered during an average run of the classifier. In this instance, the first run performed was selected as being representative of the other 99 runs conducted in the experiment. The heavy line represents the maximum fitness obtained during the run to that point, increasing as more fit solutions are encountered in subsequent generations. The line overlapping the maximum fitness to some degree represents the maximum fitness obtained during that particular generation, and in most cases is coincident with the maximum fitness obtained overall. The average fitness of the generation can be seen slightly below, but it is encouraging that the shape of the average values, despite fluctuation in individual fitness as genetic operations take place, is similar to that of the maximum fitness.

Figure 4.26 depicts the classification error encountered during trials of the system with the Fisher Iris Data Set. The area on the floor of the plot represents a classification rate of 100% for that particular run. Areas with peaks depict the rate of classification error encountered during a particular run.

Figure 4.27 illustrates the size of the best chromosome for each run during the trial. Some outliers are present, but this is to be expected of a random stochastic process such as that employed in most methods of EC. What is encouraging about this particular formation is that

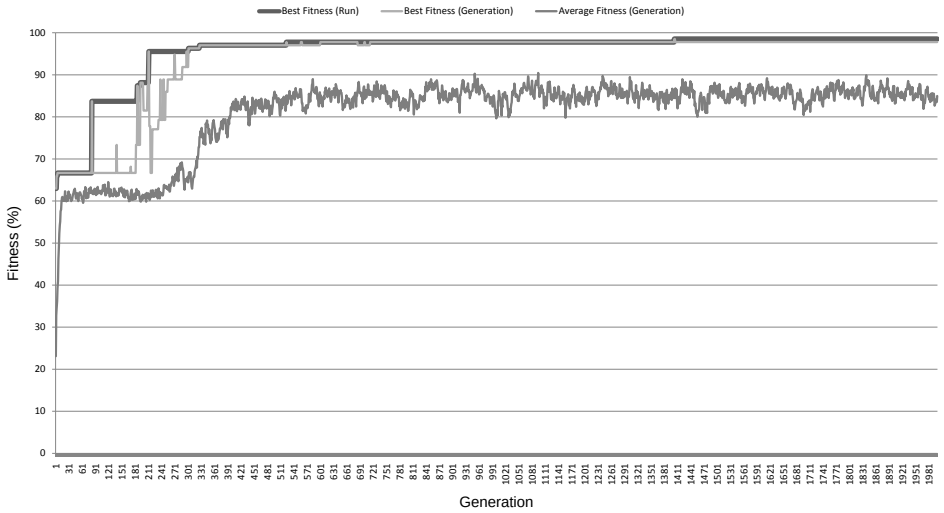

Figure 4.25: Experiment 6: Fitness values - run 0-0

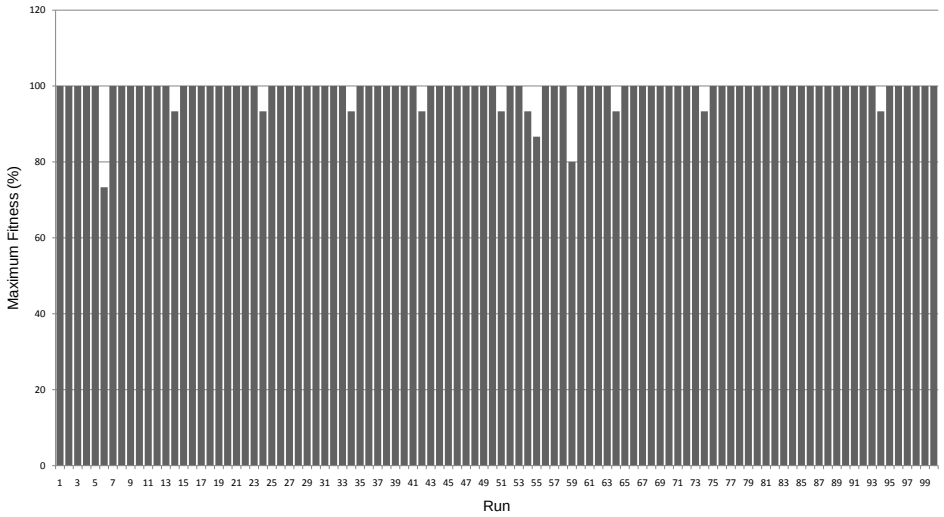

Figure 4.26: Experiment 6: Maximum fitness by run

there are no extreme outliers, and the bulk of the values lie in a band through the middle of the chart.

Table 4.3 contains the average and median values for chromosome size, as well as the standard deviation therein.

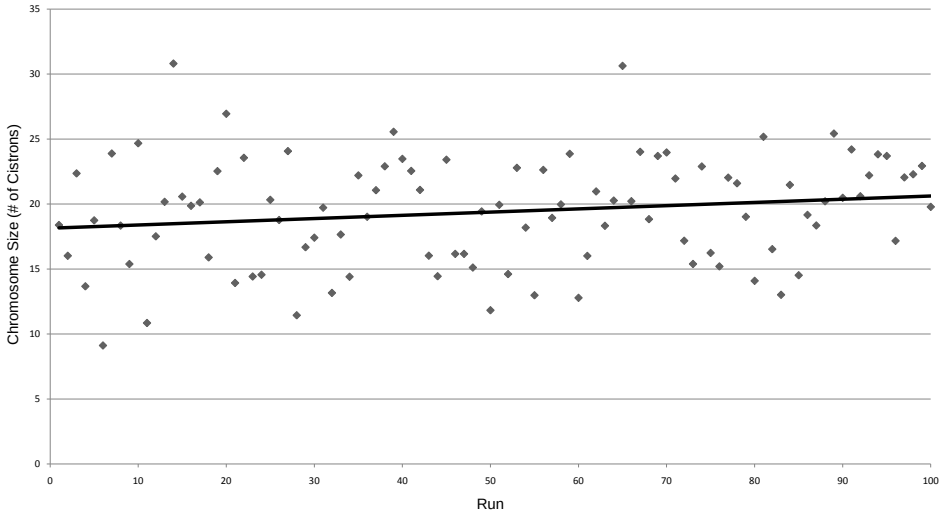

Figure 4.27: Experiment 6: Size of best solution by run

Table 4.3: Experiment 6: Chromosome size over 100 runs

	Value
Average	19.38
Median	19.82
Standard Deviation	4.18

4.3.2 Experiment 7: Wine Data Set

The results presented here for pairwise classification of the Wine data set represent a 12.875% increase in classification performance over an identical experiment performed using GEP-NN as the evolutionary framework. The results of these experiments are summarized in Table 4.4.

Table 4.4: Experiment 7: Classification results

Method	Classification Rate	Number Classified
NEXT	89.875%	(16/17.8)
GEP-NN [16]	77%	(13.72/17.8)

Figure 4.28 is illustrative of a typical run during this experiment. Note that the classification rates shown in this figure are against the training set, where the classifier achieved a classification rate of 98.13%. The results against the validation set for this run saw the same chromosome classifying 100% of records.

Figure 4.29 illustrates the maximum fitness achieved during each of the 100 runs conducted.

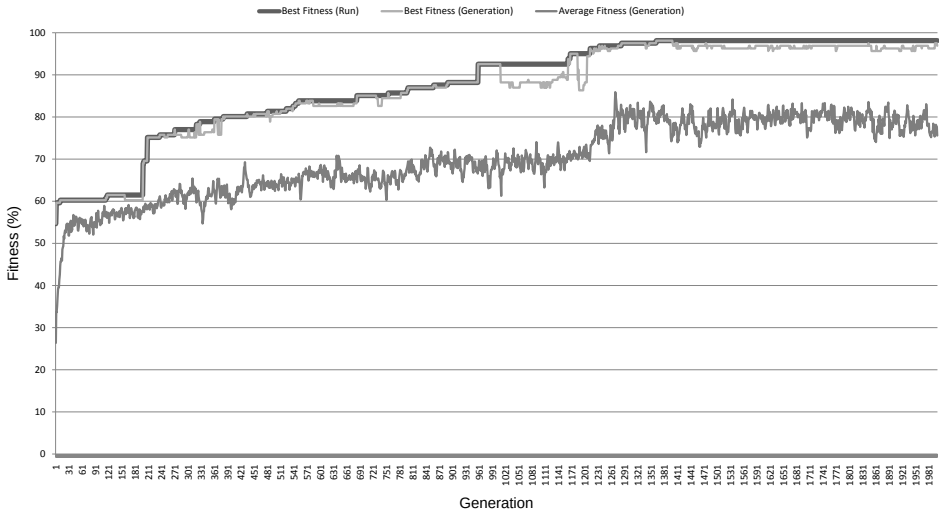

Figure 4.28: Experiment 7: Fitness values - run 3-8

The fitness levels obtained illustrate some examples of poor classification rates on individual runs, however the average classification rate was still significantly higher than that obtained testing using the GEP-NN algorithm.

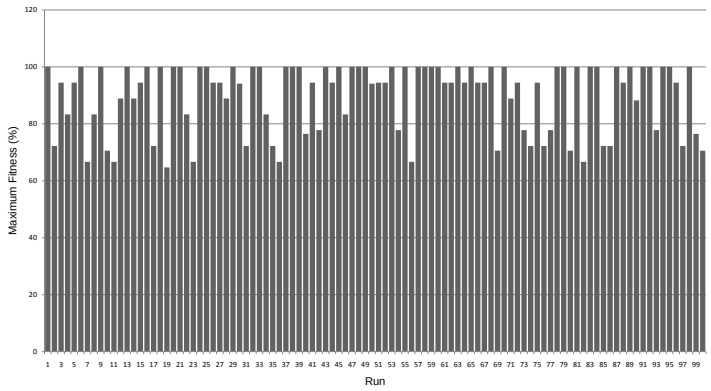

Figure 4.29: Experiment 7: Maximum fitness by run

Table 4.5: Experiment 7: Chromosome size over 100 runs

	Value
Average	13.28
Median	13.4
Standard Deviation	14.17

The chromosome sizes of the best solutions obtained during the run are illustrated in Figure 4.30 and summarized in Table 4.5. The standard deviation of the chromosome sizes found in the population is quite high, indicating that the system was experiencing difficulty in sizing the chromosomes. These values represent the average chromosome sizes of the best individuals, and despite the wide dispersal of sizes, high fitness was achieved on many solutions.

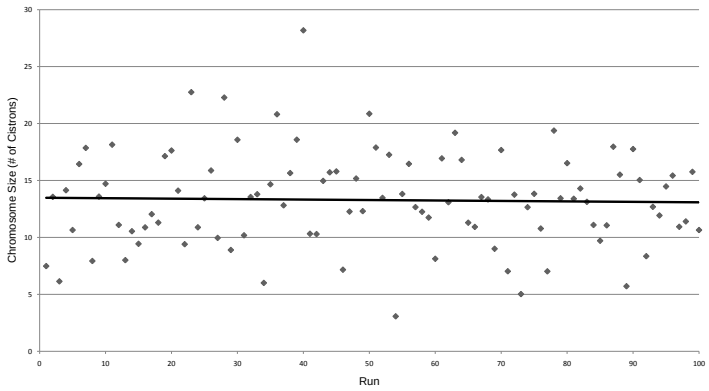

Figure 4.30: Experiment 7: Size of best solution by run

Chapter 5

Conclusion and Future Work

NEXT was developed with the aim of introducing four key characteristics to an evolutionary computation system:

- Variable-length solution representation
- Wide range of genetic operators
- Efficiency in chromosome evaluation
- Flexibility of representation

As validated through the experiments contained in this thesis, these goals were achieved through the design of a novel methodology for mathematical modeling and neuroevolution.

The experiments presented in this work showed the variable length chromosomes of NEXT to be of an asset. In several experiments, the fitness of the population increased significantly once the required chromosome size was reached. The goal in implementing a system of variable length chromosomes was to devise methods by which individuals could be properly sized without *a priori* determination of ideal chromosome size on the part of the system operator. This goal was met, and results contained herein demonstrate that the type of chromosomal bloat commonly associated with variable length systems was not evident in the results. Completely parsimonious solutions were evolved in several experiments without the aid of any external measures of parsimony pressure. On the level of obtaining the desired chromosome size without human intervention, NEXT has been successful.

The number of operators available, and indeed the number of possible operators which were not implemented in this system, represents a growth in the granularity of genetic modification

not seen in precursor systems to NEXT such as Genetic Programming or Gene Expression Programming. Operations are possible at all levels of the hierarchical structure from the codon up to wholesale modification of entire chromosome.

The method of evaluation of chromosomes in NEXT represents a decrease in the level of computational complexity over the methodology employed in GEP. The evaluation of an individual does not require explicit transformation to a phenotypic tree, nor does it require extensive traversal of tree-structures. The simple linear encoding of the chromosome, coupled with an implicit knowledge of the tree structure it represents allow for a rapid evaluation of solutions not seen in other methods.

The aim of producing a representation capable of the solution of a wide array of problems was also accomplished. Problems of three varieties were demonstrated within this thesis, including symbolic regression problems, neuroevolution geared towards logic synthesis, and finally the induction of neural networks for pairwise classification purposes. The chromosomal structure of the solutions used by all three of these problem types is nearly identical. Neuroevolution problems required only the addition of a few fields for weights and thresholds and a substitution of the function set.

Of note is the comparison of the results obtained by GEP and NEXT on identical problems. NEXT was capable of achieving 100% maximum fitness on all mathematical modeling problems attempted in comparison with GEP, however GEP saw a higher rate of success on these problems. In the neuroevolution domain, however, NEXT was more successful. In Experiment 5 (Section 4.2.1), NEXT saw 100% success rate in achieving maximum fitness, while GEP was only capable of 79%. This scenario is again repeated in Experiments 6 and 7 (Sections 4.3.1 and 4.3.2) where NEXT was able to achieve a higher rate of classification than the GEP-NN implementation of the same problem. The results for these problems were 98.8% vs 94.47% for Iris, and 89.57% vs 77% for Wine with NEXT and GEP-NN respectively. It seems possible that the structural organization of NEXT is better suited to neuroevolution than that of GEP-NN.

Particularly encouraging among the results is the frequency with which the growth of an individual by a single cistron triggered a leap in the fitness of the solution. As discussed in

Sections 4.1.1 and 4.1.3, these sorts of changes in fitness due to genotypic growth indicate that the ideas of complexification included in the design of NEXT are indeed capable of promoting the development of good solutions.

In summary, the results demonstrated in this work show that the structural design and evolutionary algorithm of NEXT were able to achieve the goals set out for its development. The variable-length representation coupled with effective operators, an efficient method of evaluation, and a flexible representation were shown to produce successful solutions to the problems demonstrated.

5.1 Future Work

While NEXT successfully met its initial goals, its development brought to light several other areas upon which future work could improve:

- Mechanism for the inclusion of constants in mathematical modeling problems.
- Development of capabilities for the induction of recurrent neural networks.
- Application of genetic operators based on statistical means.

A major limitation of NEXT in its present design is the inclusion of constants for mathematical modeling. A great number of symbolic regression problems include coefficients to the terms of functions which are not integers, and at present these are not possible to evolve within the NEXT framework. The decision not to include this capability in NEXT came down to staying within the NEXT philosophy of self-contained subtree units. GEP implements a method of evolving constants, called GEP-RNC (Random Numerical Constants), with a structure very similar to the method by which GEP-NN evolves weights and thresholds. Arrays of floating point values are stored and indexes to those arrays are included in the evolving individuals. For neuroevolution weights and thresholds, NEXT implements a system of ephemeral constants with one value stored and evolved for each vector or threshold which requires it. A variant

on this concept could be implemented in order to provide coefficient constants for symbolic regression. Experiment 4 (Section 4.1.4 suffered from poor average fitness and excessive solution size on account of the lack of constants included in NEXT. It is likely that these solutions would have been improved by the inclusion of such a mechanism.

The neural networks currently evolved by NEXT are feed-forward networks only, and are suited to only some problems of neuroevolution not requiring recurrent loops within the network. A common problem for demonstration of neuroevolution systems is the pole balancing demonstration, wherein a simulation of a cart with a hinged pole on it is balanced during motion of the cart. Recurrent networks allow such a problem to be computed with fewer inputs, and without known velocity values. The inclusion of such functionality would require alteration to the method of genotype-phenotype mapping. It seems possible that this could be accomplished through the inclusion of functions which indicate reverse links as well as the forward links implied through the function arity of neuroevolution functions. Another method which could be explored along this line is a concept of edge encoding [15] with a two-stage mapping capable of representing recurrent neural networks.

In several cases during the experiments conducted for this thesis, situations occurred where otherwise excellent experimental results were marred by poor fitness on a few individual runs. These runs, such as the example highlighted in the results of Experiment 2 (Section 4.1.2, were the result of the population becoming stuck in a situation where either the incorrect operators were spread throughout the population, or the size was not increasing as appropriate. The development of intelligent operators based upon statistical means has been a goal of the EC community at large, and would certainly benefit solutions in NEXT. An operator capable of ”noticing” these sorts of problems and taking corrective action to remove the population from these local minima would be of tremendous benefit. The results demonstrated by this system have shown that the stochastic processes employed in so many EC systems are effective, but that greater effectiveness could be achieved through statistical trending of results during runs.

Appendix A: Glossary

- **arity** - Arity refers to the number of required inputs to a function, and in the sense of expression trees, the maximum connectedness of a node.
- **bloat** - The tendency of individuals in some variable-size evolutionary algorithms to grow in an uncontrolled fashion.
- **cistron** - A unit of genetic encoding in NEXT consisting of one function and *n* terminals, representing a complete subtree.
- **codon** - The smallest unit of genetic material in either NEXT or GEP, consists of a single symbol representing a function or terminal.
- **complexification** - The notion of beginning with small chromosomes and gradually adding more genetic material to an individual through evolutionary means to more efficiently exploit the search space.
- **crossover** - A genetic operator which exchanges a unit of genetic material between two parents to produce two new offspring.
- **DGP** - *Developmental Genetic Programming*: A class of evolutionary algorithms which employ an indirect mapping and a method of genotype-phenotype mapping to permit neutral variation.
- **direct encoding** - An encoding method which makes no distinction between the genotype and phenotype of an individual, both modifying and assessing the fitness of the same structure.
- **encoding** - The EC-system-understandable rendering of a solution representation.
- **function** - Analogous to a function in programming languages, a function is some method which accepts inputs and returns an output. Arithmetic operators are an example of

typical EA functions.

- **genotype** - The genetic encoding of an individual and the entity upon which genetic operators act.
- **GPM** - *Genotype-Phenotype Mapping*: A method of translating the encoded genotype to the expressed phenotype in DGP systems.
- **indirect encoding** - An encoding method employing a GPM.
- **individual** - A candidate solution within the population of an EC system.
- **multigenic** - A solution representation where the chromosome consists of multiple genes.
- **mutation** - A genetic operator which changes one value within a genetic entity (gene, cistron, *etc.*) to a new random value.
- **neutral variation** - In an system employing an indirect encoding, neutral variation is genetic variation which currently has no effect on the phenotype but accumulates over time, and may be expressed in future with structural change.
- **non-coding region** - A region of a genotype in a system employing a GPM which is not currently expressed, *ie:* has no bearing on the phenotype at the present.
- **parsimony pressure** - A method of rewarding candidate solutions which are smaller in size to encourage the development of more compact solutions.
- **phenotype** - The entity upon which selection acts, and the expression of the genotype in DGP systems.
- **recombination** - See: *crossover*.
- **representation** - The definition by an EC methodology of what possible solutions to a problem look like.
- **search space** - The set of all possible genotypes for a given problem and encoding.

- **solution space** - The set of all possible phenotypes for a given problem and encoding.
- **terminal** - An input to a function, represented in tree-based EA systems as the leaf nodes of an expression tree.
- **TWEANN** - *Topology and Weight Evolving Artificial Neural Networks*: A class of evolutionary methodologies for complete induction of neural networks including both the topology and weights.
- **unigenic** - A solution representation where the chromosome consists of only one gene.

Bibliography

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2009.

[2] W. Banzhaf. Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. In Y. Davidor, H. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature III*, volume 866, pages 322–332. Springer-Verlag, 1994.

[3] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming- An Introduction: On the automatic evolution of computer programs and its application*. Morgan Kaufmann Publishers Inc., 1998.

[4] K. A. DeJong. *Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[5] K. A. DeJong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[6] M. Forina et. al. An extendible package for data exploration, classification and correlation, 1991.

[7] C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, pages 87–129, 2001.

[8] C. Ferreira. Designing neural networks using gene expression programming. In *Applied Soft Computing Technologies: The Challenge of Complexity*, pages 517–536. Springer-Verlag, 2006.

[9] C. Ferreira. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence, 2nd Edition*. Springer-Verlag Berlin Heidelberg, 2006.

[10] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics, 7, Part II*, pages 179–188, 1936.

[11] D. B. Fogel. Phenotypes, genotypes, and operators in evolutionary computation. In *Proceedings of the Second IEEE International Conference on Evolutionary Computation. IEEE*, pages 193–198. IEEE Press, 1995.

[12] S. Gelly, O. Teytaud, N. Bredeche, and M. Schoenauer. A statistical learning theory approach of bloat. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1783–1784. ACM, 2005.

[13] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[14] J. H. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence, Second Edition*. MIT Press., 1992.

[15] M. Łnorek J. Drchal. Tree-based indirect encodings for evolutionary development of neural networks. *Lecture Notes in Computer Science*, 5164 LNCS(PART 2):839–848, 2008.

[16] S. Johns and M.V. dos Santos. On the evolution of neural networks for pairwise classification using gene expression programming. In *GECCO 2009 Proceedings*, pages 1903–1904. ACM, 2009.

[17] R. E. Keller and W. Banzhaf. The evolution of genetic code in genetic programming. In *Proc. Genetic and Evolutionary Computation Conference*, pages 1077–1082. Morgan Kaufmann Publishers, 1999.

[18] M. Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, 1983.

[19] J.R. Koza. *Genetic Programmming: On the programming of computers by natural selection*. MIT Press, 1992.

[20] O. Lezoray and H. Cardot. A neural network architecture for data classification. *International Journal of Neural Systems*, pages 33–42, 2001.

[21] O. Lezoray and H. Cardot. Comparing combination rules of pairwise neural networks classifiers. *Neural Processing Letters*, pages 43–56, 2008.

[22] A. P. Martin. Increasing genomic complexity by gene duplication and the origin of vertebrates. *The American Naturalist*, pages 111–128, 1999.

[23] R. Poli and N. F. McPhee McPhee. Parsimony pressure made easy. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1267–1274. ACM, 2008.

[24] K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification*. PhD thesis, University of Texas at Austin, 2004.